Variable Experience Rollout: Learning Robust Skills for Embodied Rearrangement

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present Variable Experience Rollout (VER), a technique for scaling batched on-policy reinforcement learning in heterogenous environments (where different environments take vastly different times for generating rollouts). VER combines the strengths of and blurs the line between synchronous (SyncOnRL) and asynchronous (AsyncOnRL) on-policy RL methods – specifically, it learns from on-policy experience but has no synchronization points, enabling high throughput.

We find that VER leads to significant and consistent speed-ups across a broad range of embodied navigation and mobile manipulation tasks in photorealistic 3D simulation environments. Specifically, for PointGoal navigation and ObjectGoal navigation in Habitat 1.0, VER is 60-100% faster (1.6-2x speedup) over DD-PPO, the current state of art for distributed SyncOnRL. For mobile manipulation tasks (open fridge/cabinet, pick/place objects) in Habitat 2.0 VER is 150% faster (2.5x speedup) on 1 GPU and 200% faster (3x speedup) on 8 GPUs with similar or better sample efficiency. Compared to SampleFactory (AsyncOnRL), VER matches its speed on 1 GPU, and is 70% faster (1.7x speedup) on 8 GPUs with better sample efficiency.

We leverage these speed-ups to train chained skills for GeometricGoal rearrangement tasks in the Home Assistant Benchmark (HAB). We find a surprising *emergence of navigation* in skills that do not ostensible require any navigation. Specifically, the pick skill involves a robot picking an object from a table. During training the robot was always spawned close to the table and never needed to navigate. However, we find that if base movement is part of the action space, the robot learns to navigate *then* pick an object in new environments with 50% success, demonstrating surprisingly high out-of-distribution generalization.

1 Introduction

Scaling matters. Progress towards building intelligent agents that are capable of performing goal driven tasks has been driven, in part, by training large neural networks in photo-realistic 3D environments with deep reinforcement learning (RL) for (up to) billions of steps of experience [Wijmans et al., 2020, Maksymets et al., 2021, Mezghani et al., 2021, Ramakrishnan et al., 2021, Miki et al., 2022]. To enable this scale, RL systems must be able to efficiently utilize their resources, and scale to multiple machines all while maintaining sample-efficient learning.

One promising class of techniques to achieve this scale is batched on-policy RL. These methods collect experience from many environments simultaneously using the policy and update it with this cumulative experience. These methods are broadly divided into two classes: synchronous (SyncOnRL) and asynchronous (AsyncOnRL). SyncOnRL contains two potential synchronization points: first the policy is executed for the entire batch $(o_t \rightarrow a_t)_{b=1}^{B}$ (Fig. 1 A), then actions are executed in *all*

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

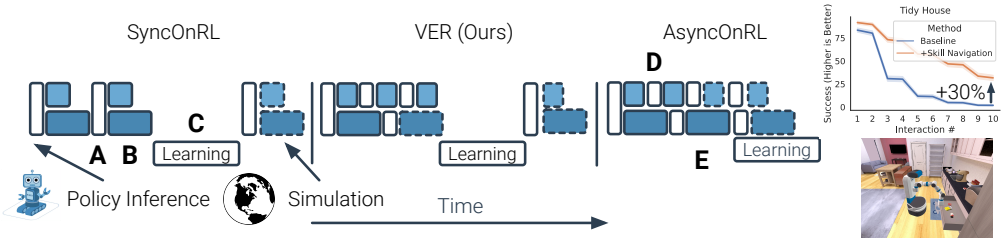

Figure 1: **(Right) RL Training Systems.** In SyncOnRL, actions are computed for all environments, then all environments are stepped. Experience collection is paused during learning. In AsyncOnRL, computing actions, stepping environments, and learning all occur without synchronization. In VER, a variable amount of experience is collected from each environment, enabling synchronous learning without the straggler effect. **(Left) skill policies** with navigation are more robust to handoff errors.

environments, $(s_t, a_t \rightarrow s_{t+1}, o_{t+1})_{b=1}^{B}$[1] (Fig. 1 B), until T steps have been collected from all N environments. This (T, N)-shaped batch of experience is used to update the policy (Fig. 1 C). The synchronization points reduce throughput due to the straggler effect, where the system spends significant (sometimes most) time idling, waiting for the slowest worker.

AsyncOnRL remove these synchronization points, thereby mitigating the straggler effect and improving throughput. Actions are taken as soon as they are computed, $a_t \rightarrow o_{t+1}$ (Fig. 1 D), the next action is computed as soon as the observation is ready, $o_t \rightarrow a_t$ (Fig. 1 E), and the policy is updated as soon as enough experience is collected. However, AsyncOnRL systems are not able to ensure that all experience has been collected by only the current policy and thus must work with *near*-policy data. This reduces sample efficiency [Liu et al., 2020]. Thus, status quo leaves us with an unpleasant tradeoff – on-policy data with low throughput or high throughput with low sample-efficiency.

In this work, we propose Variable Experience Rollout (VER). VER combines the strengths of and blurs the line between SyncOnRL and AsyncOnRL. Like SyncOnRL, VER collects experience with the current policy and then updates it. Like AsyncOnRL, VER does not have synchronization points – it computes next actions, steps environments, and updates the policy as soon as possible. The inspiration for VER comes from two key observations:

1) AsyncOnRL mitigates the straggler effect by implicitly collecting a variable amount of experience from different environments – collecting more from fast environments and less from slow ones.

2) Both SyncOnRL and AsyncOnRL collect rollouts of experience with a fixed number of steps per environment, *e.g.* (T, N)[2]. Our key insight is that an equal distribution of T steps may simplify an implementation but is *not* a requirement for RL. This insight naturally leads us to *variable experience rollouts* (VER), *i.e.* collecting rollouts with a variable number of steps across different environments.

The result is an RL system that overcomes the straggler effect *and* maintains sample-efficiency.

First we evaluate VER on well-established embodied navigation tasks using Habitat 1.0 [Savva et al., 2019]. VER trains PointGoal navigation [Anderson et al., 2018] 60% faster (1.6x speedup) than Decentralized Distributed PPO (DD-PPO) [Wijmans et al., 2020], the current state-of-the-art for distributed on-policy RL, with the same sample efficiency. For ObjectGoal navigation [Batra et al., 2020b], an active area of research, VER trains 100% faster than DD-PPO with *better* sample efficiency.

Next, we evaluate VER on the recently introduced (and significantly more challenging) GeometricGoal rearrangement tasks [Batra et al., 2020a] in Habitat 2.0 [Szot et al., 2021]. In GeoRearrange, a virtual robot is spawned in a new environment and asked to rearrange a set of objects from their initial to desired coordinates as efficiently as possible. These environments have highly variable simulation time (physics simulation time increases if the robot bumps into something) and require GPU-acceleration (for photo-realistic rendering), limiting the number of environments that can be run in parallel.

On 1 GPU, VER is 150% faster (2.5x speedup) than DD-PPO with the same sample efficiency. VER is as fast as SampleFactory [Petrenko et al., 2020], the fastest single machine AsyncOnRL, with the same sample efficiency. VER is as fast as AsyncOnRL in pure compute-efficiency, which is a high bar.

[1] Following standard notation, s_t is (PO)MDP state, a_t is the action taken, and o_t is the agent observation.
[2] In SyncOnRL these are N unique environments. In AsyncOnRL they may not be unique.

We then combine VER with DD-PPO to scale to multiple GPUs. VER achieves better scaling than DD-PPO, achieving a 6.7x speed-up on 8 GPUs (vs. 6x for DD-PPO) due to lower variance in experience collection time between GPU-workers. Due to highly efficient multi-GPU scaling, VER is 70% faster than SampleFactory on 8 GPUs and more sample-efficient.

We leverage these SysML contributions to study open research questions posed in prior work. Specifically, we train RL policies for mobile manipulation skills (navigation, pick, place, *etc*.) and chain them via a task planner. Szot et al. [2021] called this approach TP-SRL and identified a critical 'handoff problem' – downstream skills are set up for failure by small errors made by upstream skills (*e.g*., the robot stopping a bit too far to pick up an object).

We demonstrate a number of surprising findings when TP-SRL is scaled via VER. Most importantly, we find the *emergence of navigation* when skills that do not ostensibly require navigation (*e.g*. pick) are trained with navigation actions enabled. In principle, pick and place policies do not *need* to navigate during training since the objects are always in arm's reach, but in practice they learn to navigate to recover from their mistakes and this results in strong out-of-distribution test-time generalization. Specifically, TP-SRL *without a navigation skill* achieves 50% success on NavPick and 20% success on a NavPickNavPlace task simply because the pick and place skills have learned to navigate (sometimes across the room). TP-SRL with a navigation skill performs even stronger: 90% on NavPickNavPlace and 32% on 5 successive NavPickNavPlaces (called Tidy House in Szot et al. [2021]), which are +32% and +30% absolute improvements over Szot et al. [2021]. Prepare Groceries and Set Table, which both require interaction with articulated receptacles (fridge, drawer), remain as open problems and are the next frontiers.

2 VER: Variable Experience Rollout

The key challenge that any large-batch RL technique needs to address is variability of simulation time for the environments in a batch. There are two primary sources of this variability: action-level and episode-level. The amount of time needed to simulate an action within an environment varies depending on the the specific action, the state of the robot, and the environment (*e.g*. simulating the robot navigating on a clear floor is much faster than simulating the robot's arm colliding with objects). The amount of time needed to simulate an entire episode also varies environment to environment irrespective of action-level variability (*e.g*. rendering images can take longer for visually-complex scenes, simulating physics can take longer for scenes with a large number of objects).

2.1 Action-Level Straggler Mitigation

We mitigate the action-level straggler effect by applying the experience collection technique of AsyncOnRL to SyncOnRL. There are two components, policy workers and actor workers.

Actor workers receive the next action in their environment and step the environment ($a_t \rightarrow o_{t+1}$). They write the outputs of the environment (observations, reward, and whether or not the episode terminated) into pre-allocated CPU shared memory for consumption by the policy worker.

Policy workers receive steps of experience from actor workers. They perform inference with the current policy to select the next action and send it to the actor worker using pre-allocated CPU shared memory. After inference policy workers store experience for learning in shared GPU memory. policy workers use dynamic batching and only process steps after a threshold or a time-limit is reached.

This experience collection technique is similar to that of HTS-RL [Liu et al., 2020] (SyncOnRL) and SampleFactory Petrenko et al. [2020] (AsyncOnRL). Unlike both, we do not overlap experience collection with learning. This has various system benefits, including reducing GPU memory usage and reducing GPU driver context switching, which improves usage.

2.2 Environment-Level Straggler Mitigation

In both SyncOnRL and AsyncOnRL, the experience used for learning consists of N sets of T steps of experience, an (T, N)-shaped batch. In SyncOnRL these N sets are all collected with the current policy, this the environment-level straggler effect. AsyncOnRL mitigates this by relaxing the constraint that experience must be strictly on-policy, and thereby implicitly changes the experience collection rate for each environment.

We instead relax the constraint that we must use N sets of T steps. Specifically, VER collects $T \times N$ steps of experience from N environments without a constraint on how many steps of experience are collected from each environment (see Fig. 2 (A)). This explicitly varies the experience collection

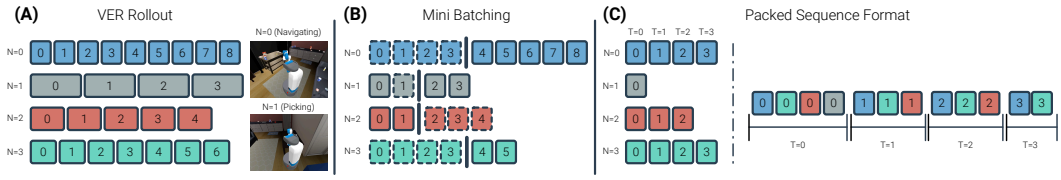

Figure 2: **(A) `VER` collects a variable amount of experience from each environment.** The length of each step represents the time taken to collect it. **(B) `VER` mini-batch.** The solid bars denote episode boundaries. The steps selected for the first mini-batch have a dashed border. **(C) The `PackedSequence` data format** represents a set of sequences with variable length in a linear buffer such that all elements from each timestep area next to one-another in memory.

rate for each environment – in effect, collecting more experience from environments that are fast to simulate. Compared to `SyncOnRL`, `VER` does not suffer from the environment-level straggler effect. Compared to `AsyncOnRL`, `VER` does not require correction for stale data.

One subtle design choice is the following – when `VER` finishes a $(T \times N)$ rollout collection, there will be (slow) environments that haven't completed simulation yet. Instead of discarding that data, we choose to collect this experience in the *next* rollout. This experience is at most 1 policy-update old, contains at most $N - 1$ steps, and we find this choice leads to speed gains without any sample-efficiency loss.

Learning mini-batch creation. When training a recurrent policy, we must create batches of experience with sequences for back-propagation-through-time (BPTT). Normally, the (T, N) shaped batch of experience is divided between the mini-batches along the environment (N) axis. A similar procedure would result in mini-batches of different sizes for `VER`. We instead divide the sequences of experience (splitting sequences if needed).

First, note that there are two reasons for the start of a new sequence of experience. Rollout starts (Fig. 2 (B), step 0) and episode starts (Fig. 2 (B), a step after a bar). These two boundaries types are independent – episodes can start at any arbitrary step within the rollout. Thus when we collect experience from N environments, we will have $K \geq N$ sequences to divide between the mini-batches. We distributed these K sequences between the mini-batches. We use a greedy algorithm that seeks to 1) maximizes the number of environments in each mini-batch, 2) minimizes the number of sequences that need to be cut, and 3) cuts the shortest sequences first. See Fig. 2 (B) for an example.

Batching computation for learning. The mini-batches constructed from the algorithm above have sequences with variable length. To batch the computation of these sequences we use cuDNN's `PackedSequence` data model. This data model represents a set of sequences with variable length (Fig. 2 (C) left) such that all elements for a given time-step are in a contiguous block of memory (Fig. 2 (C) right) – this enables efficient batched computation on each time-step. Further, it uses a contiguous block of memory for all elements. This enables batched computation on all elements for network components that don't have a temporal aspect, such the visual encoder.

During experience collection we write experience into a linear buffer. After experience collection we arrange each mini-batch as a `PackedSequence`. This takes less than 10 milliseconds (per rollout phase); orders of magnitude less than experience collection ($\sim$3s) or learning ($\sim$1.5s).

2.3 Multiple GPUs

We apply the decentralized distributed method from DD-PPO [Wijmans et al., 2020] to scale our system to multiple GPUs. We adapt and build upon the straggler preemption method presented therein. During experience collection, we track when each step of experience was added to the learning buffer. We then compute the average time to collect a step of experience for each environment. Finally, we compute the optimal number of steps of experience to collect for that rollout and use it to decide when to preempt on the next rollout. We find this method to be more robust than the fixed threshold used in DD-PPO as it is able to adapt to changes in simulation time throughout learning. Unlike DD-PPO we don't reduce the learning batch-size for workers that don't collect the full amount of experience. We fill the rollout with experience from the previous rollout. This improves performance when using cuDNN as this remembers what kernels worked best for each batch-size. When we copy experience from a previous rollout, we don't recompute the return. In effect, we perform extra epochs of PPO on this experience. This comes with no effective computation

cost since the result is that all GPUs have the same batch size instead of some having a smaller batch size. In practice we never see the faction of previous experience exceed 1%.

3 Embodied Navigation: Benchmarking

First, we benchmark VER on the embodied navigation [Anderson et al., 2018] tasks in Habitat 1.0 [Savva et al., 2019] – PointNav [Anderson et al., 2018] and ObjectNav [Batra et al., 2020b]. Our goal here is simply to show training speed-up in well-studied tasks (and in the case of PointNav, a well-saturated task with no room left for accuracy improvements). We present accuracy improvements and in-depth analysis on rearrangements tasks in Section 6.

Task	DD-PPO	VER
PointGoalNav	3065	5325
ObjectNav	1021	2019

Table 1: **Navigation Tasks.** VER is 60%-100% faster than DD-PPO.

For both tasks, we use standard architectures from Habitat Baselines [Savva et al., 2019, Wijmans et al., 2020] – the ResNet18 encoder and a 2 layer LSTM. Following Ye et al. [2020, 2021], we add Action Conditional Contrastive Coding [Guo et al., 2018], using the hyper-parameters from Guo et al. [2020].

PointNav. We train PointNav agents with an RGB camera on the HM3D dataset [Ramakrishnan et al., 2021] for 1.85 billion steps of experience on 8 GPUs. We examine RGB as this the more challenging version of the task and thus we expect it to be more sensitive to possible differences in the training system. VER trains agents 60% faster than DD-PPO (Table 1) with similar sample efficiency (Fig. 3).

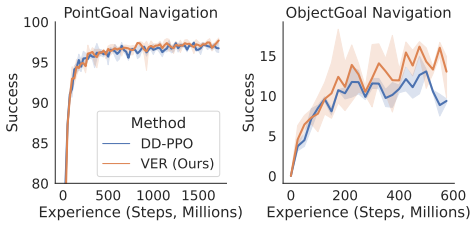

Figure 3: **Validation performance** on ObjectNav and PointNav. Shading is a 95% confidence interval over 3 seeds.

ObjectNav. We train ObjectNav agents with RGB and Depth cameras on the MP3D+[Chang et al., 2017] dataset for 600 million steps of experience on 8 GPUs. VER trains agents 100% faster than DD-PPO with better sample efficiency. There are two effects that enable better sample efficiency with VER. First, we perform additional epochs of PPO on experience from the last rollout when a GPU-worker is preempted. Second, the variable experience rollout mechanism results in a natural curriculum. Environments are often faster to simulate when they are easier (*i.e.* a smaller home), so more experience will be collected in these easier cases.

4 Embodied Rearrangement: Task, Agent, and Training

Next, we use VER to study the recently introduced (and more challenging) GeometricGoal rearrangement rearrangement tasks [Batra et al., 2020a] in Habitat 2.0 [Szot et al., 2021].

Task. In GeoRearrange, an agent is initialized in a novel unseen environment and tasked with rearranging its environment. The task is specified as a set of coordinate pair $\{(\text{Pose}_{\text{Initial}}, \text{Pose}_{\text{Final}})\}_{o=1}^{O}$. The agent must bring each object at $\text{Pose}_{\text{Initial}}$ to $\text{Pose}_{\text{Final}}$. HAB consists of 3 scenarios of increasing difficulty: Tidy House, Prepare Groceries, and Set Table.

Simulation. We use the Habitat simulator [Savva et al., 2019, Szot et al., 2021]. The robot policy operates at 30 Hz and physics is simulated at 120 Hz.

Agent. The agent is embodied as a Fetch robot with a 7-DOF arm. The arm is controlled via joint velocities. At every time step the policy predicts a delta in motor position for each of the 7 joints in the arm. We find joint velocity control equally easy to learn but faster to simulate than the end-effector control used in Szot et al. [2021]. The arm is equipped with a suction gripper. The agent must control the arm such that the gripper is in contact with the object to grasp and then activate the gripper. The object is dropped once the gripper is deactivated. This is more realistic than the 'magic' grasp action used in Szot et al. [2021]. The robot base is controlled by the policy commanding a desired linear speed and angular velocity. The robot is equipped with a singular Depth camera attached to its head. The policy models $a_t \sim \pi(\cdot \mid s_{t-1})$ instead of $a_t \sim \pi(\cdot \mid s_t)$. This is both more realistic and enables physics and rendering to be overlapped [Szot et al., 2021].

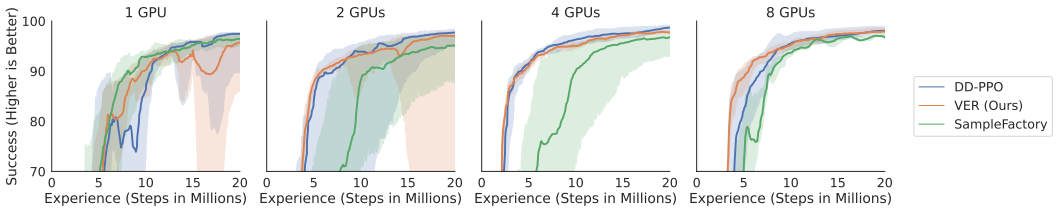

Figure 4: **Training sample efficiency** on Open Fridge. VER has similar sample efficiency as DD-PPO (SyncOnRL). SampleFactory (AsyncOnRL) has similar sample efficiency with 1 GPU but this reduces as policy lag increases with more GPUs. The shaded region is a 95% bootstrapped confidence interval over 5 seeds. We use interquartile mean (IQM) as our summary statistic [Agarwal et al., 2021].

We build upon the TaskPlanning-SkillRL (TP-SRL) method proposed in Szot et al. [2021]. TP-SRL is a hierarchical method for GeoRearrange that decomposes the task into a series of skills – Navigate, Pick, Place, and $\{\text{Open, Close}\} \times \{\text{Cabinet, Fridge}\}$. Skills are controlled via a skill-policy (learned with RL) and chained together via a task planner. One of the key challenges is the 'handoff problem' – downstream skills are setup for failure due to slight errors made by the upstream skill. We give all skill policies access to navigation actions to allow them to correct for these errors.

Architecture. Our skill policies all share the same architecture. We use ResNet18 [He et al., 2016] to process the 128×128 visual input. Following Wijmans et al. [2020], we reduce with width of the network by half and use GroupNorm [Wu and He, 2018]. We also apply some of the recent advancements from ConvNeXt [Liu et al., 2022]. We use patch-ify stem, dedicated down-sample stages, layer scale, and dilated convolutions (this mimics larger kernel convolutions without increasing computation). The visual embedding is then combined with the state observations and previous action and then processed with a 2-layer LSTM [Hochreiter and Schmidhuber, 1997]. The output of the LSTM is used to predict the action distribution and value function. Actions are sampled multivariate Gaussian distribution with a diagonal covariance matrix.

Training. We train agents using VER and Proximal Policy Optimization [Schulman et al., 2017] with Generalized Advantage Estimation [Schulman et al., 2016]. We use an entropy constraint with a learned coefficient [Haarnoja et al., 2018] as we find this to be more stable given our diverse set of skills. Formally, let $\mathcal{H}(\pi)$ be the entropy of the current policy, we then minimize $-[[\alpha]]_{\text{sg}}\mathcal{H}(\pi) + \alpha(\lambda + [[\mathcal{H}(\pi)]]_{\text{sg}})$ where $[[\cdot]]_{\text{sg}}$ is the stop gradient operator. We set λ to zero for all tasks. We use the Adam optimizer [Kingma and Ba, 2015] with an initial learning rate of 2.5×10^{-4} and decay the learning rate to zero using a cosine schedule. To correct for the biased sampling of VER, we use truncated importance sampling weighting [Espeholt et al., 2018] with a maximum of 1.0.

5 Embodied Rearrangement: Benchmarking

We examine VER along two axes: 1) training throughput – the number of samples of experience per second (SPS) the system collects and learns from, 2) sample efficiency. In this section, we report benchmarking results for the open-fridge policy because this task involves a complex interaction of the robot with an articulated object (the fridge) and represents a challenging case for the training system due to large variability in physics time. In Section 6, we analyze the task performance, which requires all skills. All systems use 16 environments per GPU.

5.1 System throughout

We compare the provided HTS-RL [Liu et al., 2020] implementation[3] with our re-implementation, VER minus variable experience rollouts and overlapped experience collection and learning (NoVER+Overlap). NoVER+Overlap is 110% faster than HTS-RL (Table 2). Key differences in our implementation are fast userspace mutexes (futexes) in shared memory (vs. spin locks), pre-allocated pinned memory for CPU to GPU transfers (vs. allocating for each transfer), and GPU shared

RNN	HTS-RL	NoVER+Overlap	NoVER	VER
×	242	506	501	620
✓	-	450	462	590

Table 2: **HTS-RL comparison.** Mean system throughput (SPS) over 1 million training steps. HTS-RL does not support training recurrent policies. Hardware: Nvidia 2080 Ti with 16 CPUs.

[3] https://github.com/IouJenLiu/HTS-RL

memory to send experience from policy workers to learn and weights from learner to policy workers (vs. CPU shared memory). There has no significant change in system SPS if we remove overlapped experience collection and learning (Table 2, NoVER+Overlap vs. NoVER), while doing so simplifies the implementation, and reduces GPU memory usage (∼2 GB). We note that overlapped experience collection and learning does have uses, *i.e.* for CPU simulation or policies with significant CPU components, but it isn't necessary when both the policy and simulator make heavy use of the GPU.

NoVER is a 'steel-manned' baseline for VER and benefits from all our micro-optimizations. The only difference is whether we collect a variable experience rollout or not. VER is 30% faster than NoVER. Both in our HTS-RL comparison (Table 2) and SyncOnRL and AsyncOnRL comparison (Table 3).

VER is 150% faster than DD-PPO [Wijmans et al., 2020] (Table 3), the fastest SyncOnRL implementation for training agents with recurrent policies. DD-PPO has no mechanism to mitigate the action-level or episode-level straggler effects. Early in training when the agent is not interacting with objects, DD-PPO has similar throughput as VER (Table 3, Max). Once the robot begins bumping objects the throughput of DD-PPO reduces by 150% compared to 20% for VER (Table 3, Mean vs. Max).

	DD-PPO		NoVER		VER (Ours)		SampleFactory	
GPUs	Mean	Max	Mean	Max	Mean	Max	Mean	Max
1	174	442	327	428	428	534	427	517
2	283	696	592	786	716	945	804	1022
4	468	1337	1097	1601	1432	1915	1286	1568
8	1066	2754	2216	3438	2861	3829	1662	1842

Table 3: **SyncOnRL, VER, and AsyncOnRL benchmarking.** Mean and max system throughput (SPS) over 20 million training steps. Hardware: Nvidia DGX-1 with Tesla V100 with 10 CPUs per GPU.

VER closes the gap to AsyncOnRL. On 1 GPU VER is as fast as SampleFactory [Petrenko et al., 2020], the fastest single machine AsyncOnRL. Intuitively AsyncOnRL should be an upper-bound on performance – it never stops collecting experience while VER does. However this doesn't take into account the realities of hardware. Recall that we are training an agent with a large visual encoder. This means that updating the parameters of the agent takes a large amount of time (∼150ms per mini-batch of size 1024 on a V100). Further, Habitat uses the GPU for rendering. The use of the GPU for both rendering and learning simultaneously results in context switches by the GPU driver. In SampleFactory, learning time and experience collection time are double that of VER.

Multi-GPU scaling. VER better multi-GPU scaling than DD-PPO, achieving a 6.7x speed-up on 8 GPUs compared to 6x. DD-PPO introduces a third straggler effect, the GPU-worker-level, which their straggler preemption seeks to mitigate. While that method does work, it isn't perfect and scaling would be better if it wasn't needed. Rollouts in VER are lower variance, which improves scaling. On 2 GPUs, SampleFactory is 12% faster than VER. In this case, one GPU is used for learning+inference and the other is used for rendering. This creates a nice division of work and doesn't result in costly context switches. On 4 and 8 GPUs however, the single GPU used for learning in SampleFactory is the bottleneck and VER has higher throughput (nearly 100% faster 8 GPUs). While it is possible to implement multi-GPU learning for AsyncOnRL to overcome this problem, it is left to the user to balance the number of GPUs used for experience collection and learning. VER automatically balances GPU-time used for experience and learning.

5.2 Sample Efficiently

Next we examine sample efficiency of the training systems. VER has either identical (1, 2, and 4 GPUs), Fig. 4) or better sample efficiency (8 GPUs) than DD-PPO. Interestingly, SampleFactory has similar sample efficiency with 1 GPU (Fig. 4) and is more stable. This is because reducing the number of GPUs also reduces the batch-size and PPO is known to not be batch-size invariant [Hilton et al., 2021]. We believe the stale data serves a similar function to PPO-EMWA [Hilton et al., 2021], which uses an exponential moving average of the policy weights for the importance sampling terms. On 2, 4, and 8 GPUs, VER has better sample efficiency.

6 Embodied Rearrangement: Analysis of Learned Skills

Next we examine the performance of TP-SRL on the Home Assistant Benchmark (HAB) [Szot et al., 2021]. We examine both skill policies trained with the full action space and the limited and per-skill

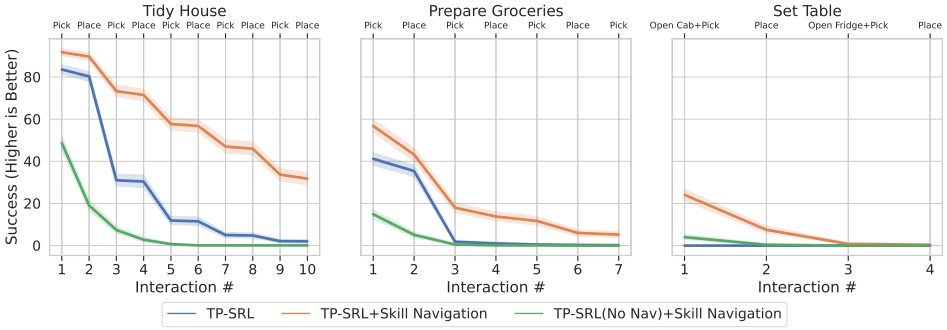

Figure 5: **HAB Performance** on the Tidy House, Prepare Groceries, and Set Table scenarios. Skill policies with navigation (TP-SRL+All navigation) outperform skill policies without navigation (TP-SRL) despite not strictly needing this ability. Further, we find that these skill policies have learned *emergent* navigation (TP-SRL(NoNav)+All navigation).

specific action space used in Szot et al. [2021]. Each skills is trained with VER for 500 million steps of experience on 8 GPUs. This takes less than 2 days per skill.

6.1 Performance on HAB

We find that using skills with navigation improves performance on the full task but does not change performance on the skill's train task (*e.g.* pick achieves 90% Success regardless). Skills with navigation are able to overcome errors of the navigation policy. Fig. 5 shows smaller drops in performance between every interaction (there is a navigation between each interaction). This is impactful after place as the navigation policy tends to make more errors when navigating to the next location after place. On Tidy House, full task performance improves from 2% Success to 32%.

On Prepare Groceries (which requires picking/placing from/into the fridge) and Set Table (which requires opening the fridge or cabinet and then picking from it) performance improves slightly. 0% to 5% on Prepare Groceries and 0% to 7.5% on the first Pick+Place on Set Table. Both these tasks remain as open problems and the next frontier.

6.2 Emergent Navigation

Next we examine the extent that the skill policies are able to use navigation to correct for out-of-distribution initial locations. We examine an extreme case and construct a version of the TP-SRL agent that omits the Nav skill – TP-SRL(NoNav). In this agent *all* navigation is done by skill policies that ostensibly never needed to navigate during training.

We find *emergent navigation* in both the pick and place policies. On Tidy house, the pick skills successfully *navigates* and picks up the object 50% of the time on Tidy House (Fig. 5). The place policy successfully navigates and places the object at the desired location 40% of the time (Fig. 5 20/50). The limiting factor for the place policy's navigation performance is that it often mistakenly drops the object when navigating, not that it doesn't know where to go or how to avoid obstacles. This is on-par with the TaskPlanning+SensePlanAct (TP-SPA) classical baseline and significantly better than the MonolithicRL baseline in Szot et al. [2021].

The pick and place policy were trained on a task that requires no navigation but both are capable of navigation. We provided examples of both the training task for pick and place, and TP-SRL(NoNav) on Tidy House in the supplementary materials.

On Prepare Groceries and Set Table, the navigation performance of these policies is worse. Prepare Groceries requires picking from the fridge while Set Table requires opening the cabinet and then picking. This requires navigation that does not accidentally bump the fridge door/cabinet draw and close it. Performance is non-zero in both these scenarios however, indicting that the skill policies are capable of navigating even in these scenarios; albeit less successfully than in Tidy House.

We hypothesize that the pick and place polices learned navigation because this was useful *early* in training. Early in training the policies have yet to learn that only minimal navigation is needed to

complete the task. Therefore the policy will sometimes cause itself to move away from the pick object/place location and will navigate back. Navigation is then not forgotten as the policy converges.

We examined the behavior of a pick policy early in training and found that it does tend to move away from the object it needs to pick up and sometimes moves back. Although the magnitude of navigation is small and navigation quite infrequent, so the degree of generalization is high.

This result, and the higher performance on HAB, highlights that it may not always be beneficial to remove 'unneeded' actions. Presumably Szot et al. [2021] removed navigation where possible to improve sample efficiency and/or training throughput – in our own experiments, training without navigation both improves sample efficiency and throughput. However, by enabling navigation and allowing the agent to learn that only minimal navigation is needed itself, we arrived upon emergent navigation and improved both full task performance.

7 Related Work

AsyncOnRL systems provide high-throughput on-policy reinforcement learning [Espeholt et al., 2018, Petrenko et al., 2020]. However, they have reduced sample efficiency as they must correct for near-policy, or 'stale', data. Few support multi-GPU learning and, when they do, the user must manually balance compute between learning and experience collection [Espeholt et al., 2020]. VER achieves the same throughput on 1-GPU while learning with on-policy data, has better sample efficiency, supports multi-GPU learning, and automatically balances compute between learning and simulation.

SyncOnRL Systems. Closely related to our work, HTS-RL [Liu et al., 2020] also use the same experience collection techniques as AsyncOnRL to mitigate the action-level straggler effect. We propose a novel mechanism, variable experience rollouts, to mitigate the episode-level straggler effect. We use and build upon Decentralized Distributed PPO (DD-PPO) [Wijmans et al., 2020], which proposed a distributed multi-GPU method based upon data parallelism [Hillis and Steele Jr, 1986].

Batched simulators simulate multiple agents (in multiple environments) and are responsible for their own parallelization [Shacklett et al., 2021, Petrenko et al., 2021, Freeman et al., 2021, Makoviychuk et al., 2021]. While these systems offer impressive performance, none currently support a benchmark like HAB (which combines physics and photo-realism) nor do any offer the flexibility of Habitat, AI2Thor [Kolve et al., 2017], or ThreeDWorld [Gan et al., 2020]. VER enables researchers to first explore promising directions using existing simulators and then build batched simulators.

8 Societal Impact, Limitations, and Conclusion

Our main application result is trained using the ReplicaCAD dataset [Szot et al., 2021], which is limited to only US apartments, and this may have negative societal impacts for deployed assistants. VER was designed and evaluated for tasks with both GPU simulation and large neural networks. For taks with CPU simulation and smaller networks, we expect it to improve upon SyncOnRL but it may have less throughput than AsyncOnRL and overlapping experience collection and learning would likely be beneficial[4]. The TP-SRL agent we build upon requires oracle knowledge, *i.e.* that the cabinet must be opened before picking.

We have presented Variable Experience Rollout (VER). VER combines the strengths of and blurs the line between SyncOnRL and AsyncOnRL. Its trains agents for embodied navigation tasks in Habitat 1.0 60%-100% faster (1.6x to 2x speedup) than DD-PPO with similar or better sample efficiency – saving 19.2 GPU-days on PointNav and 28 GPU-day for ObjectNav. On the recently introduced (and more challenging) embodied rearrangement tasks in Habitat 2.0, VER trains agents 150% faster than DD-PPO and is fast as SampleFactory (AsyncOnRL) on 1 GPU. On 8 GPUs, VER is 180% faster than DD-PPO and 70% faster than SampleFactory with better sample efficiency – saving 32 GPU-days per skill vs. DD-PPO and 11.2 GPU-days vs. SampleFactory. We use VER to study rearrangement. We find the *emergence of navigation* in policies that ostensibly require no navigation when given access to navigation actions. This results in strong progress on Tidy House (+30% success). This results highlights that it may not always be advantageous to limit a policy's action space.

[4]There is fundamental to VER that prevents this, but it isn't implemented in our provided implementation.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021. 6

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 2, 5

Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, Manolis Savva, and Hao Su. Rearrangement: A challenge for embodied ai. In *arXiv preprint arXiv:2011.01975*, 2020a. 2, 5

Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects, 2020b. 2, 5

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *International Conference on 3D Vision (3DV)*, 2017. License: http://kaldir.vc.in.tum.de/matterport/MP_TOS.pdf. 5

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018. 6, 9

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 9

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021. 9

Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020. 9

Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A Pires, and Rémi Munos. Neural predictive belief representations. *arXiv preprint arXiv:1811.06407*, 2018. 5

Zhaohan Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-Bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3875–3886. PMLR, 2020. 5

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018. 6

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6

W Daniel Hillis and Guy L Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29(12): 1170–1183, 1986. 9

Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization. *arXiv preprint arXiv:2110.00641*, 2021. 7

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. 6

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. 6

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. 9

Iou-Jen Liu, Raymond Yeh, and Alexander Schwing. High-throughput synchronous deep rl. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020. 2, 3, 6, 9

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022. 6

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. 9

Oleksandr Maksymets, Vincent Cartillier, Aaron Gokaslan, Erik Wijmans, Wojciech Galuba, Stefan Lee, and Dhruv Batra. Thda: Treasure hunt data augmentation for semantic navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15374–15383, 2021. 1

Lina Mezghani, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksandr Maksymets, Dhruv Batra, Piotr Bojanowski, and Karteek Alahari. Memory-augmented reinforcement learning for image-goal navigation. *arXiv preprint arXiv:2101.05181*, 2021. 1

Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7 (62):eabk2822, 2022. 1

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 International Conference on Machine Learning (ICML)*, pages 7652–7662. PMLR, 2020. 2, 3, 7, 9

Aleksei Petrenko, Erik Wijmans, Brennan Shacklett, and Vladlen Koltun. Megaverse: Simulating embodied agents at one million experiences per second. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 8556–8566. PMLR, 2021. 9

Santhosh K Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alex Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, et al. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. *Neural Information Processing Systems – Benchmarks and Datasets*, 2021. 1, 5

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2019. 2, 5

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016. 6

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 6

Brennan Shacklett, Erik Wijmans, Aleksei Petrenko, Manolis Savva, Dhruv Batra, Vladlen Koltun, and Kayvon Fatahalian. Large batch simulation for deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. 9

Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun,

Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 2, 3, 5, 6, 7, 8, 9

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 1, 2, 4, 5, 6, 7, 9

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2018. 6

Joel Ye, Dhruv Batra, Erik Wijmans†, and Abhishek Das†. Auxiliary tasks speed up learning pointgoal navigation. *Conference on Robot Learning (CoRL) (In submission)*, 2020. 5

Joel Ye, Dhruv Batra, Abhishek Das, and Erik Wijmans. Auxiliary tasks and exploration enable objectnav. *arXiv preprint arXiv:2104.04112*, 2021. 5

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 8
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See supplementary materials
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We use 95% bootstrapped confidence intervals.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] .
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See supplementary materials
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] See supplementary materials
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] See supplementary materials
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]