VER: Scaling On-Policy RL Leads to the Emergence of Navigation in Embodied Rearrangement

Erik Wijmans[1,2] **Irfan Essa**[1,3] **Dhruv Batra**[2,1]
[1] Georgia Institute of Technology [2] Meta AI [3] Google Atlanta
{etw,irfan,dbatra}@gatech.edu

Abstract

We present Variable Experience Rollout (VER), a technique for efficiently scaling batched on-policy reinforcement learning in heterogenous environments (where different environments take vastly different times for generating rollouts) to many GPUs residing on, potentially, many machines. VER combines the strengths of and blurs the line between synchronous and asynchronous on-policy RL methods (SyncOnRL and AsyncOnRL, respectively). Specifically, it learns from on-policy experience (like SyncOnRL) and has no synchronization points (like AsyncOnRL), 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) than DD-PPO, the current state of art for distributed SyncOnRL, with similar sample efficiency. 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 170% faster (2.7x speedup) on 8 GPUs than DD-PPO. Compared to SampleFactory (the current state-of-the-art 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.

Code: github.com/facebookresearch/habitat-lab

1 Introduction

Scaling matters. Progress towards building embodied 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 the available resources (*e.g.* GPUs), 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 (N) environments simultaneously using the policy and update it with this cumulative experience. They are broadly divided into two classes: synchronous (SyncOnRL)

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

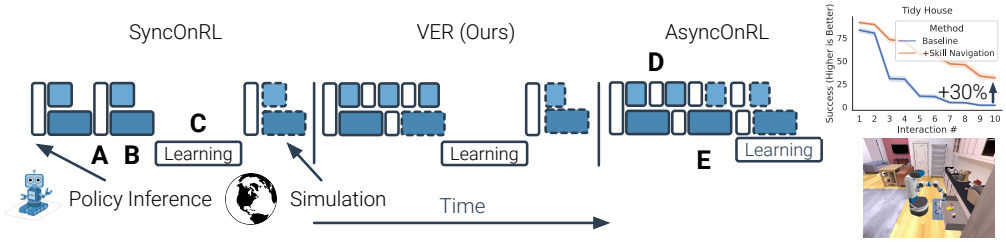

Figure 1: **(Left) RL Training Systems.** In SyncOnRL, actions are computed for all environments, then all environments are stepped. Experience collection is paused during learning. In AsyncOnRL, computing actions, stepping environments, and learning all occur without synchronization. In VER, a variable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

and asynchronous (AsyncOnRL). SyncOnRL contains two synchronization points: first the policy is executed for the entire batch $(o_t \rightarrow a_t)_{b=1}^{B}$ [1] (Fig. 1 A), then actions are executed in *all* environments, $(s_t, a_t \rightarrow s_{t+1}, o_{t+1})_{b=1}^{B}$ (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). Synchronization reduces throughput as the system spends significant (sometimes the most) time waiting for the slowest environment to finish. This is the straggler effect [Petrini et al., 2003, Dean and Ghemawat, 2004].

AsyncOnRL removes 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 consume *near*-policy data. This reduces sample efficiency [Liu et al., 2020]. Thus, status quo leaves us with an unpleasant tradeoff – high sample-efficiency 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 each environment – more from fast-to-simulate environments and less from slow ones.

2) Both SyncOnRL and AsyncOnRL use a fixed rollout length, T steps of experience. Our key insight is that while a fixed rollout length may simplify an implementation, it is *not* a requirement for RL.

These two key observations naturally lead us to *variable experience rollout* (VER), *i.e.* collecting rollouts with a variable number of steps. VER adjusts the rollout length for each environment based on its simulation speed. It explicitly collects more experience from fast-to-simulate environments and less from slow ones (Fig. 1). The result is an RL system that overcomes the straggler effect *and* maintains sample-efficiency by learning from on-policy data.

VER focuses on efficiently utilizing a single GPU. To enable efficient scaling to multiple GPUs, we combine VER with the decentralized distributed method proposed in [Wijmans et al., 2020].

First, we evaluate VER on well-established embodied navigation tasks using Habitat 1.0 [Savva et al., 2019] on 8 GPUs. VER trains PointGoal navigation [Anderson et al., 2018] 60% faster 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 is 100% faster than DD-PPO with (slightly) 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. These environments have highly variable simulation time (physics simulation

[1] Following standard notation, s_t is (PO)MDP state, a_t is the action taken, and o_t is the agent observation.

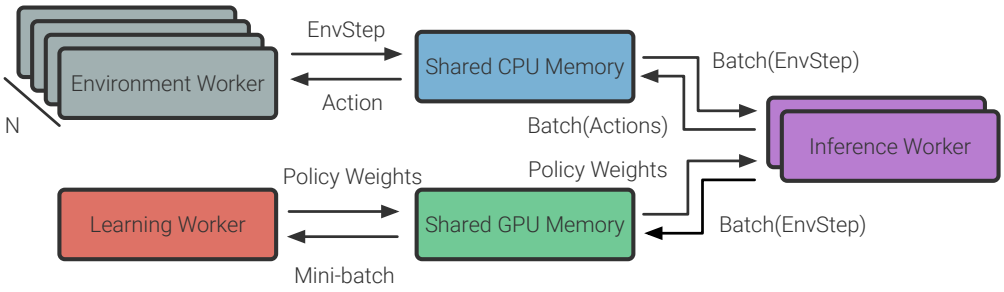

Figure 2: **VER system architecture.** Environment workers receive actions to simulate and return the result of that environment step (EnvStep). Inference workers receive batches experience from environment workers. They return the new action to take to environment workers and write the experience into GPU shared memory for learning.

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 state-of-the-art AsyncOnRL, with the same sample efficiency. VER is as fast as AsyncOnRL in pure throughput; this is a surprisingly strong result. AsyncOnRL never stops collecting experience and should, in theory, be a strict upper bound on performance. VER is able to match AsyncOnRL for environments that heavily utilize the GPU for rendering, like Habitat. In AsyncOnRL, learning, inference, and rendering contend the GPU which reduces throughput. In VER, inference and rendering contend for the GPU while learning does not.

On 8 GPUs, VER achieves better scaling than DD-PPO, achieving a 6.7x speed-up (vs. 6x for DD-PPO) due to lower variance in experience collection time between GPU-workers. Due to this efficient multi-GPU scaling, VER is 70% faster (1.7x speedup) than SampleFactory on 8 GPUs and has better sample efficiency as it learns from on-policy data.

Finally, we leverage these SysML contributions to study open research questions posed in prior work. Specifically, we train RL policies for mobile manipulation skills (Navigate, 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 Pick skill failing because the navigation skill stopped the robot a bit too far from the 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 Navigate 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], respectively. Prepare Groceries and Set Table, which both require interaction with articulated receptacles (fridge, drawer), remain as open problems (5% and 0% Success, respectively) and are the next frontiers.

2 VER: Variable Experience Rollout

The key challenge that any batched on-policy 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 takes longer for visually-complex scenes, simulating physics takes longer for scenes with a large number of objects).

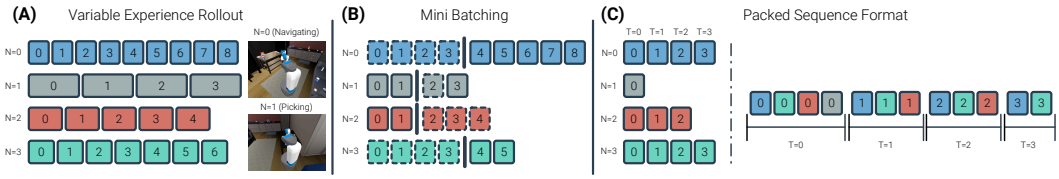

Figure 3: **(A) VER** collects a variable amount of experience from each environment. The length of each step represents the time taken to collect it. **(B) VER mini-batch.** The solid bars denote episode boundaries. The steps selected for the first mini-batch have a dashed border. **(C) The PackedSequence 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.

2.1 Action-Level Straggler Mitigation

We mitigate the action-level straggler effect by applying the experience collection method of AsyncOnRL to SyncOnRL. We represent this visually in Fig. 2 and describe it in text bellow.

Environment workers receive the next action and step the environment, EnvStep, *e.g.* $s_t, a_t \rightarrow s_{t+1}, o_{t+1}, r_t$. They write the outputs of the environment (observations, reward, *etc.*) into pre-allocated CPU shared memory for consumption by inference workers.

Inference workers receive batches of steps of experience from environment workers. They perform inference with the current policy to select the next action and send it to the environment worker using pre-allocated CPU shared memory. After inference they store experience for learning in shared GPU memory. Inference workers use dynamic batching and perform inference on all outstanding inference requests instead of waiting for a fixed number of requests to arrive[2]. This allows us to leverage the benefits of batching without introducing synchronization points between environment workers.

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 contention. More details are available in Appendix A.

2.2 Environment-Level Straggler Mitigation

In both SyncOnRL and AsyncOnRL, the data used for learning consists of N rollouts of equal-T steps of experience, an (T, N)-shaped batch. In SyncOnRL these N sets are all collected with the current policy, this leads to 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.

Variable Experience Rollout (VER). We instead relax the constraint that we must use N rollouts of equal-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. This explicitly varies the experience collection rate for each environment – in effect, collecting more experience from environments that are fast to simulate. Consider the 4 environments shows in Fig. 3A. The length of the each step representation the wall-clock time taken to collect it, some steps are fast, some are slow. VER collects more experience from environment 0 as it is fastest to step and less from 1, the slowest.

Learning mini-batch creation. VER is designed with recurrent policies in mind because memory is key in long-range and partially observable tasks like HAB. When training recurrent policies, we must create mini-batches of experience with sequences for back-propagation-through-time. Normally B mini-batches are constructed by spitting the N environments' experience into B $(T, {}^{N}/_{B})$-sized mini-batches. A similar procedure would result in mini-batches of different sizes with VER. This would harm optimization because learning rate and optimization mini-batch size are intertwined and automatically adjusting the learning rate is an open question [Goyal et al., 2017, You et al., 2020].

To under VER's mini-batching, first note that there are two reasons for the start of a new sequence of experience: rollout starts (Fig. 3B, step 0) and episode starts (Fig. 3B, a step after a bar). These

[2]In practice we introduce both a minimum and maximum number of requests to prevent under-utilization of compute and over-utilization of memory.

two boundaries types are independent – episodes can end at any arbitrary step within the rollout and then that environment will reset and start a new episode. 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 randomly order the sequences, then the first $T \times N / B$ steps are the first mini-batch, the next $T \times N / B$ to the second, *etc*. See Fig. 3B 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 variable-length sequences (Fig. 3C left) such that all sequence-elements at a time-step are contiguous in memory (Fig. 3C right) – enabling batched computation on each time-step for components with a temporal dependence, *e.g*. the RNN – and that all elements across all time-steps are also contiguous – enabling batched computation across *all* time-steps for network components that don't have a temporal dependence, *e.g*. the visual encoder.

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

Learning method. The experience and mini-batches generated from `VER` are well-suited for use with RL methods that use on-policy data [Sutton and Barto, 1992]. We use Proximal Policy Optimization (PPO) [Schulman et al., 2017] as it is known to work well for embodied AI tasks and recurrent policies.

Inflight actions One subtle design choice is the following – when `VER` finishes a $T \times N$ experience 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. We find this choice leads to speed gains without any sample-efficiency loss.

2.3 Multiple GPUs

We leverage the decentralized distributed training architecture from Wijmans et al. [2020] to scale `VER` to multiple GPUs (residing on a single or multiple nodes). In this architecture, each GPU both collects experience and learns from that experience. During learning, gradients from each GPU-worker are averaged with an `AllReduce` operation. This is a synchronous operation and thus introduces a GPU-worker-level straggler effect. Wijmans et al. [2020] mitigate this effect by preempting stragglers – stopping collecting experience early and proceeding to learning – after a fixed number of GPU-workers have finished experience collection.

We instead approximate the optimal preemption for each experience collection phase. Given the learning time, LT, and a function Time(S) that returns the time needed to collect S steps of experience, the optimal number of steps S to collect before preempting stragglers is

$$\max_{S} \frac{S}{\text{Time}(S) + \text{LT}}, \quad \text{s.t.} \quad S \leq T \cdot N \cdot \#\text{GPUs}. \tag{1}$$

For LT, we record the time from the last iteration as this doesn't change between iterations. We approximate Time(S) using the average step time to receive a step of experience from each environment (this includes both inference time and simulation time) from the previous rollout. This will be a poor approximation in some circumstances but we find it to work well in practice.

We improve upon sample efficiency of DD-PPO by filling the preempted rollouts with experience from the previous rollout. We perform extra epochs of PPO on this now 'stale' data instead of correcting for the off-policy return as we find this simpler and effective. This comes with no effective computation cost since the maximum batch size per GPU is unchanged.

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-ups 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 challenging rearrangement tasks in Section 6.

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].

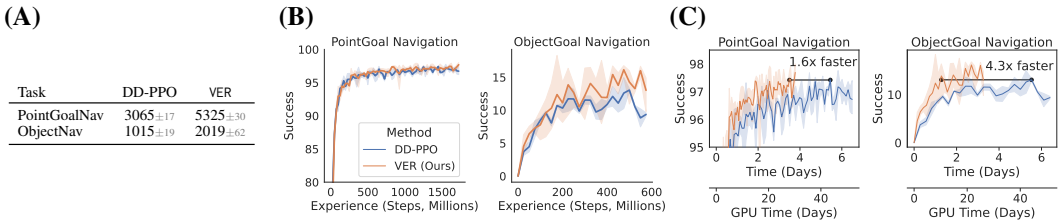

Figure 4: **(A) Navigation Tasks** training steps per second on 8 GPUs. VER is 60%-100% faster than DD-PPO. **(B) Sample efficiency** on ObjectNav and PointNav performance (validation success). VER has similar or slightly better sample efficiency than DD-PPO, indicating that performance is not negatively impacted by the non-uniform sampling of experience from environments. Shading is a 95% confidence interval over 3 seeds. **(C) Compute efficiency** on ObjectNav and PointNav performance (validation success). To reach the maximum success achieved by DD-PPO, 97.4% on PointNav and 13.0% on ObjectNav, VER uses 1.6x less compute on PointNav (saving 16 GPU-days) and 4.3x less compute on ObjectNav (saving 33.4 GPU-days).

PointNav. We train PointNav agents with one RGB camera on the HM3D dataset [Ramakrishnan et al., 2021] for 1.85 billion steps of experience on 8 GPUs. We study the RGB setting (and not Depth) because this is the more challenging version of the task and thus we expect it to be more sensitive to possible differences in the training system. 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. On 8 GPUs, VER trains agents 60% faster than DD-PPO, from 3065 SPS to 5325 SPS (Fig. 4A), with similar sample efficiency (Fig. 4B).

ObjectNav. We train ObjectNav agents with one RGB and one Depth camera 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, 1021 to 2019 (Fig. 4A), with slightly better sample efficiency (Fig. 4B). Due to improved throughput and sample efficiency, VER uses 4.3x less compute than DD-PPO to reach the same success (Fig. 4C). 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 an unknown environment and tasked with rearranging objects in its environment. The task is specified as a set of coordinate pairs $\{(\text{Pose}_{\text{Initial}}, \text{Pose}_{\text{Final}})\}_{o=1}^{O}$. The agent must bring each object at $\text{Pose}_{\text{Initial}}$ to $\text{Pose}_{\text{Final}}$ where Pose is the initial or desired center-of-mass location for the object(s). We use the Home Assistant Benchmark (HAB) which consists of 3 scenarios of increasing difficulty: Tidy House, Prepare Groceries, and Set Table.

In Tidy House, the agent is tasked with moving 5 objects from their initial locations to their final locations. The objects are rarely in containers (*i.e.* fridge or cabinet drawer) and when they are, the containers are already opened. In Prepare Groceries, the agent must move 3 objects from the kitchen counter into the open fridge (or open fridge to counter). This stresses picking and placing in the fridge, which is challenging. In Set Table, the agent must move 1 object from the closed kitchen cabinet drawer to the table and 1 object from the closed fridge to the table. This requires opening the cabinet and fridge, and picking from these challenging receptacles.

Simulation. We use the Habitat simulator with the ReplicaCAD Dataset [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 joint 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

	DD-PPO		NoVER		VER (Ours)		SampleFactory	
GPUs	Mean	Max	Mean	Max	Mean	Max	Mean	Max
1	174$\pm$7	442$\pm$9	327$\pm$7	428$\pm$11	428$\pm$5	534$\pm$7	427$\pm$5	517$\pm$3
2	283$\pm$23	696$\pm$24	592$\pm$5	786$\pm$11	716$\pm$32	945$\pm$29	804$\pm$15	1022$\pm$0
4	468$\pm$21	1337$\pm$34	1097$\pm$30	1601$\pm$39	1432$\pm$10	1915$\pm$13	1286$\pm$6	1568$\pm$26
8	1066$\pm$84	2754$\pm$156	2216$\pm$60	3438$\pm$94	2861$\pm$21	3829$\pm$23	1662$\pm$4	1842$\pm$0

Table 1: **SyncOnRL, VER, and AsyncOnRL benchmarking.** Mean/max system throughput (SPS) over 20 million training steps. VER is 150% faster than DD-PPO on 1 GPU and 170% faster on 8 GPUs. Hardware: Tesla V100(s) with 10 CPUs per GPU.

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 (navigation) is controlled by the policy commanding a desired linear speed and angular velocity. The robot is equipped with one `Depth` camera attached to its head, proprioceptive sensors that provide the joint positions of its arm, and a `GPS+Compass` sensor that provides its heading and location relative to its initial location. The policy models $a_t \sim \pi(\cdot \mid s_{t-1})$ instead of $a_t \sim \pi(\cdot \mid s_t)$. This is more realistic and enables physics and rendering to be overlapped [Szot 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 $\{$`Open`, `Close` $\}$ $\times$ $\{$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. All skill policies share the same architecture. We use ResNet18 [He et al., 2016] to process the 128$\times$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 a patch-ify stem, dedicated down-sample stages, layer scale [Touvron et al., 2021], and dilated convolutions [Yu and Koltun, 2015] (this mimics larger kernel convolutions without increasing computation). The visual embedding is then combined with the proprioceptive 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 from independent Gaussian distributions.

Training. We train agents using VER and PPO [Schulman et al., 2017] with Generalized Advantage Estimation [Schulman et al., 2016]. We use a minimum entropy constraint with a learned coefficient [Haarnoja et al., 2018] as we find this to be more stable given our diverse set of skills than a fixed coefficient. Formally, let $\mathcal{H}(\pi)$ be the entropy of the policy, we then minimize $\alpha(\lambda - [[\mathcal{H}(\pi)]]_{\text{sg}}) - [[\alpha]]_{\text{sg}}\mathcal{H}(\pi)$ where $[[\cdot]]_{\text{sg}}$ is the stop gradient operator. We set the target entropy, 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 it to zero with a cosine schedule. To correct for biased sampling in VER, we use truncated importance sampling weighting [Espeholt et al., 2018] with a maximum of 1.0.

5 Embodied Rearrangement: Benchmarking

In this section, we benchmark VER for training open-fridge policies because this task involves 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. For all systems, we set the number of environments, N, to 16 per GPU.

5.1 System throughout

VER is 150% faster than DD-PPO (Table 1); an even larger difference than in the simpler navigation tasks we studied before. DD-PPO has no mechanism to mitigate the action-level or episode-level straggler effects. In their absence, DD-PPO has similar throughput as VER (Table 1, Max). Under these effects, DD-PPO's throughput reduces 150% compared to 20% for VER (Table 1, Mean vs. Max).

Variable experience rollouts are effective. We compare VER with VER minus variable experience rollouts (NoVER). NoVER is a 'steel-manned' baseline for VER and benefits from all our micro-optimizations. VER is 30% faster than NoVER (Table 1).

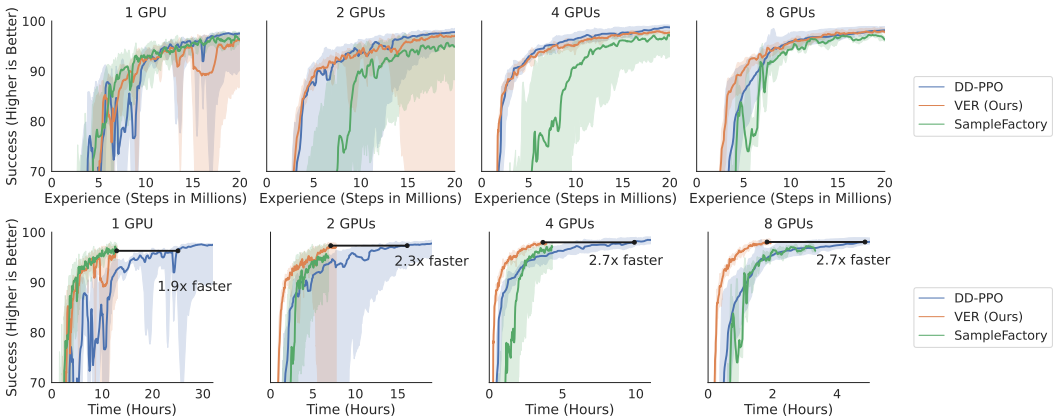

Figure 5: **Training efficiency** on Open Fridge. VER has similar sample efficiency as DD-PPO (SyncOnRL). SampleFactory (AsyncOnRL) has similar sample efficiency with 1 GPU but this reduces as policy lag increases with more GPUs. On 8 GPUs, VER uses 2.7x less compute than DD-PPO to reach the same performance. This saves 32-days of GPU-time, per skill, for the full training schedule of 500 million frames. 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].

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 a strict 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 GPU driver contention. In SampleFactory, learning time and experience collection time are roughly double that of VER.

Multi-GPU scaling. VER has better multi-GPU scaling than DD-PPO, achieving a 6.7x speed-up on 8 GPUs compared to 6x. The rollout collection time in VER is lower variance, which improves scaling. On 2 GPUs, SampleFactory is 12% faster than VER. Here 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 GPU contention. 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 on 8 GPUs). While it is possible to implement multi-GPU learning for AsyncOnRL, it is left to the user to balance the number of GPUs used for experience collection and learning and this balance is static. VER automatically balances GPU-time used for experience and learning continuously.

5.2 Sample and Compute Efficiency

Next we examine sample efficiency of the training systems. On 1 GPU, VER has slightly worse sample efficiency than DD-PPO (Fig. 5). We suspect that this could be fixed by hyper-parameter choice. On 2 and 4 GPUs, VER has identical sample efficiency and better sample efficiency on 8 GPUs, Fig. 5 Interestingly, SampleFactory has similar sample efficiency with 1 GPU (Fig. 5) 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-EWMA [Hilton et al., 2021], which uses an exponential moving average of the policy weights for the trust region. On 2, 4, and 8 GPUs, VER has better sample efficiency than SampleFactory. Combing the findings of throughput and sample efficiency, we find that VER always achieves a given performance threshold in the least amount of training time (outright or a tie) (Fig. 5 Lower).

6 Embodied Rearrangement: Analysis of Learned Skills

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, per-skill

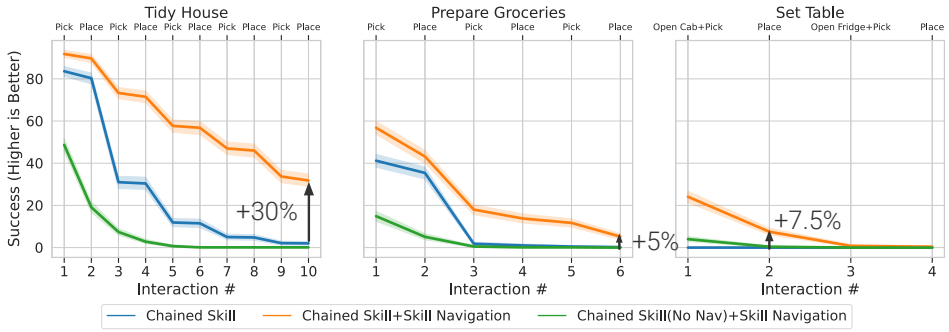

Figure 6: **HAB Performance** on the Tidy House, Prepare Groceries, and Set Table scenarios. Skill policies with navigation (TP-SRL+Skill Navigation) outperform skill policies without navigation (TP-SRL) despite not strictly needing this ability. Further, we find that these skill policies have learned *emergent* navigation. TP-SRL(NoNav)+Skill Navigation achieves 50% success on NavPick (Tidy House) and 20% success on NavPickNavPlace (Tidy House) despite all navigation being performed by skills policies that ostensibly don't need to navigate.

specific action space used in Szot et al. [2021]. Each skill 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 skills with navigation actions improve performance on the full task but does not change performance on the skill's train task (*e.g.* `Pick` achieves 90% success with and without during training). Fig. 6 shows smaller drops in performance between every interaction (there is a navigation between each interaction) demonstrating that skills with navigation effectively correct for handoff errors. 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/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 if the skill policies are able to navigate to correct for highly out-of-distribution initial location. We examine TP-SRL(NoNav), which that omits the navigation skill. 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. TP-SRL(NoNav) achieves 50% success on NavPick (Fig. 6, Tidy House interaction 1), and 20% success on NavPickNavPlace (Fig. 6, Tidy House interaction 2). The latter 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` policies were trained on tasks 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 (-34% Success and -45% Success on the first interaction, respectively). Prepare Groceries requires picking from the fridge, which is challenging and requires navigation that doesn't accidentally close the door. Set Table requires opening the cabinet and then picking, which introduces an additional OpenCab skill and requires more precise navigation and picking from `Pick`. Performance is non-zero (15% and 4% on the first interaction, respectively) in both scenarios; 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 quite infrequent, so the degree of generalization is high.

This result, and higher performance on HAB, highlights that it may not always be beneficial to remove 'unneeded' actions. Szot et al. [2021] removed navigation where possible to improve sample efficiency and training throughput[3]. Our experiments corroborate this; training without navigation improves both sample efficiency and training speed. However, by enabling navigation and allowing the agent to learn how to (not) use it, we arrived upon emergent navigation and improved HAB performance.

7 Related Work

`AsyncOnRL` methods 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`. HTS-RL [Liu et al., 2020] also use the experience collection techniques as `AsyncOnRL` to mitigate the action-level straggler effect. Unfortunately inefficiencies in the provided implementation prevent a meaningful direct comparison and hide the full effectiveness of this technique. In Appendix E we show that our re-implementation is 110% faster (2.1x speedup) and thus instead compare to the stronger baseline of No`VER` in the main text. We propose a novel mechanism, variable experience rollouts, to mitigate the episode-level straggler effect and thereby close the gap to `AsyncOnRL`. We use and build upon Decentralized Distributed PPO (DD-PPO) [Wijmans et al., 2020], which proposed a distributed multi-GPU method based on data parallelism [Hillis and Steele Jr, 1986].

Batched simulators simulate multiple agents (in multiple environments) simultaneously 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 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 with the knowledged gained from their findings.

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 tasks 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. Our implementation supports overlapping learning and collecting 1 rollout, but more overlap may be beneficial. The TP-SRL agent we build upon requires oracle knowledge, *e.g.* 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 sample efficiency – saving 19.2 GPU-days on PointNav and 28 GPU-day for ObjectNav per seed in our experiments. 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 vs. DD-PPO and 11.2 GPU-days vs. SampleFactory per skill in our experiments. 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) and shows that it may not always be advantageous to remove 'unneeded actions'.

[3]Personal correspondence with authors.

Acknowledgements. The Georgia Tech effort was supported in part by NSF, ONR YIP, and ARO PECASE. EW is supported in part by an ARCS fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government, or any sponsor.

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. 8

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, 6

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

Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015. 17

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. 6, 17

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004. 2

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. 7, 10

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. 10

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. 10

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. 10

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 4

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. 16

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. 7

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. 7

W Daniel Hillis and Guy L Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29(12): 1170–1183, 1986. 10

Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization. *arXiv preprint arXiv:2110.00641*, 2021. 8

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. 7

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. 7, 16

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. 10

Iou-Jen Liu, Raymond Yeh, and Alexander Schwing. High-throughput synchronous deep rl. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020. 2, 4, 10, 16

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. 7

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. 10

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. 3, 4, 8, 10

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. 10

Fabrizio Petrini, Darren J Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q. In *SC'03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, pages 55–55. IEEE, 2003. 2

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, 6, 17

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, 6

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. 7, 16

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 5, 7

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. 10

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1992. 5

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, 6, 7, 8, 9, 10, 15, 17

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 32–42, 2021. 7

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, 5, 7, 10

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2018. 7

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

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 4

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. 7

A Systems Considerations

A.1 Memory benefits of not overlapping experience collection and learning

VER does not overlap experience collection and learning. This allows the main process to act as an inference worker during experience collection and a learner during learning, saving 1 CUDA context ($\sim$1GB of GPU memory).

Unlike AsyncOnRL, VER is able to use GPU shared memory to transfer experience from inference workers to learner. This is because VER uses (at least) half the GPU memory for storing rollouts. AsyncOnRL must maintain at least 1 set of rollout buffers for storing the rollout currently being used for learning an another set to store the next rollout being collected. In practice, they maintain more than this minimum 2x to enable faster training.

A.2 Graphics Processing Units (GPUs)

The application of graphics processing units (GPUs) to training neural networks has led to many advances in artificial intelligence. In reinforcement learning, GPUs are used for policy inference, learning the policy, and rendering visual observations like Depth or RGB. Given the importance of GPUs in reinforcement learning, there are several important attributes of them to understand.

1) GPUs are not optimized for multiple processes using them concurrently. They are unable to efficiently execute multiple processes at once (if at all). While sometimes concurrent use is the only option, a system should whenever possible have a single process that has exclusive use of the GPU and uses it fully. 2) GPU drivers have different compute modes for graphics (*i.e.* rendering) and compute (*i.e.* neural network inference). Current GPU drivers have to perform an expensive context switch whenever alternating between these modes. Thus a system should minimize the number of times the GPU must switch between compute and graphics. 3) The compute mode CUDA context for process and each CUDA context requires a large amount of memory. Thus a system should use the minimal number of CUDA contexts possible.

B Skill Deployment

We use the same skills and deployement procedure as Szot et al. [2021] with two changes to the navigation skill: 1) We have a dedicated stop action for navigation instead of using $a_t = \mathbf{0}$ as stop. We find this to be more robust. 2) We add a force penalty to training, this makes the navigation skill less likely to bump into things by accident. However, this also makes it stop immediately if it ever starts in contact with the environment. During evaluation, we mask its prediction of stop if the target object is more than 2 meters away. This information is part of its sensor suite, so this does not require any privileged information.

On Set Table, we add an additional stage to the planner that calls navigation again after open cabinet as the open cabinet skill with base movement tends to move away from the cabinet.

C Videos

In the supplement, we include the following videos.

Video 1 (1-tp-srl-no-nav.mp4) This is an example of TP-SRL(NoNav)+All Base Movement on Tidy House. The objects to pick are have a white wireframe box drawn around them and at their place locations there is another copy of that object. The grey sphere on the ground denotes where the navigation skill was trained to navigate to. This information is not shown to the policy.

The policy makes errors and picks up the wrong object

Video 2 (2-tp-srl.mp4) This is an example of TP-SRL+All Base Movement on Tidy House.

Video 3 (3-pick-*.mp4) Examples of the pick policy on its training task. Note the difference in spawn distance between this and Video 1.

Video 3 (3-place-*.mp4) Examples of the place policy on its training task.

Optimizer	Adam [Kingma and Ba, 2015]
Initial Learning Rate	2.5×10^{-4}
Final Learning Rate	0
Decay Schedule	Cosine
Total Training Steps	500 Million
PPO Epochs	3
Mini-batches per Epoch	2
PPO Clip	0.2
Initial Entropy Coefficient	10^{-3}
Entropy Coefficient Bounds	$[10^{-4}, 1.0]$
Target Entropy	0.0
Value Loss Coefficient	0.5
Clipped Value Loss	No
Normalized Advantage	No
GAE Parameter [Schulman et al., 2016] (λ)	0.95
Discount Factor (γ)	0.99
VER Important Sampling	Yes
Max VER IS	1.0
Number of Environments (N) – per GPU	16
Experience per rollout ($T \times N$) – per GPU	128×16
Number of GPUs	8

Table A1: Hyperparameters

RNN	HTS-RL (Provided)	HTS-RL (Ours)	NoVER	VER
×	242	506	501	620
✓	-	450	462	590

Table A2: **HTS-RL comparison.** Mean system throughput (SPS) over 1 million training steps. HTS-RL (Provided) does not support training recurrent policies. Hardware: Nvidia 2080 Ti with 16 CPUs.

D Hyperparamaters

For CPC|A, we use the hyperparameters from Guo et al. [2020].

The hyperparameters for our rearrangement skills are in Table A1.

For the benchmark comparisons, we set the rollout length T to 128 for methods that require this, that way all methods collect the same amount of experience per rollout. For SampleFactory, we use a batch size of 1024 on 1 GPU, 2048 on 2, and 4096 on 4 and 8 (this the GPU memory limit). We do no use a learned entropy coefficient for benchmarking as not all frameworks support this. We use a fixed value of 10^{-4} as this works well for Open Fridge.

E HTS-RL Comparison

HTS-RL [Liu et al., 2020] uses the sampling method of AsyncOnRL to collect experience for SyncOnRL (the same as NoVER and VER). It further uses delayed gradients to enable overlapped experience and learning. Unfortunately the provided implementation[4] has inefficiencies that limit its throughput.

To demonstrate this, we add the same style of overlapped experience collection and learning to NoVER and compare the provided HTS-RL implementation, HTS-RL (Provided), with our re-implementation, HTS-RL (Ours). Our implementation 110% faster than the provided implementation (Table A2). Key differences in our implementation are fast userspace mutexes (futexes) in shared memory for synchronization (vs. spin locks), pre-allocated pinned memory for CPU to GPU transfers (vs. allocating

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

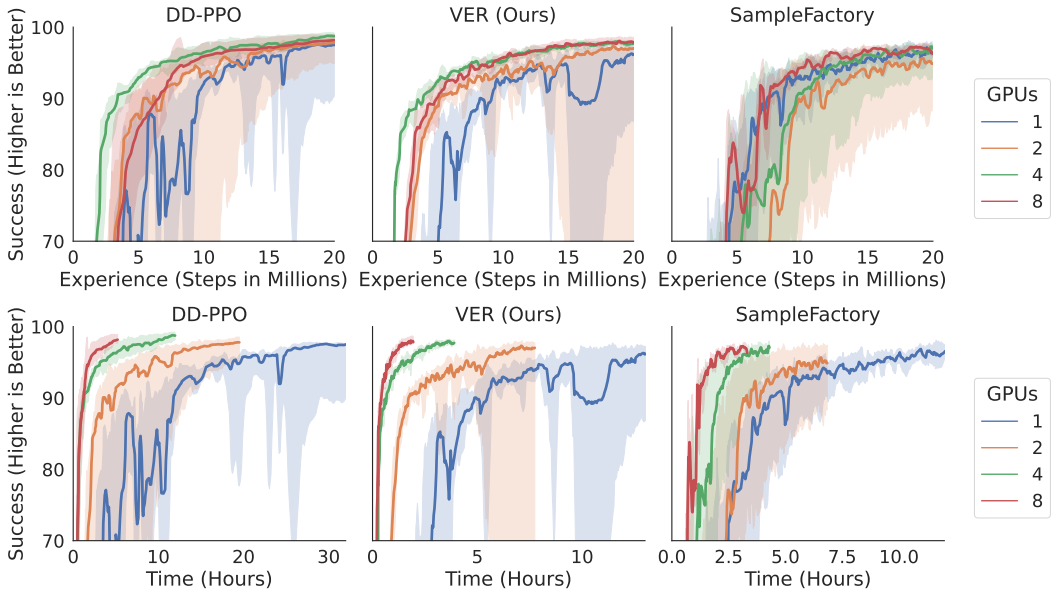

Figure A1: Sample efficiency and time-to-sample for each system on a varying number of GPUs. Even with the reduction of sample efficiency when increasing the number of GPUs, increasing the the number of GPUs always reduces the time to reach convergence.

for each transfer), and GPU shared memory to send experience from inference workers to learn and weights from learner to inference workers (vs. CPU shared memory).

We also find that given the efficiency of our implementation, there is no significant change in system SPS if we remove overlapped experience collection and learning (Table A2, HTS-RL (Ours) vs. NoVER) for Habitat-style tasks. Removing this reduces GPU memory usage (∼2 GB in our experiments, this will be larger for larger networks). 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.

F Datasets

ReplicaCAD [Szot et al., 2021] – Creative Commons Attribution 4.0 International (CC BY 4.0) license. This dataset was artist constructed.

YCB dataset [Calli et al., 2015] – Creative Commons Attribution 4.0 International (CC BY 4.0). This dataset contains 3D scans of generic objects. There is no PII.

Matterport 3D [Chang et al., 2017] – http://kaldir.vc.in.tum.de/matterport/MP_TOS.pdf. Consent was given by the owners of the space.

Habitat Matterport Research Dataset [Ramakrishnan et al., 2021] – https://matterport.com/matterport-end-user-license-agreement-academic-use-model-data. Consent was given by the owners of the space and any PII was blurred.

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 Appendix F
   (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 Appendix F
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] See Appendix F
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]