Quantization for Sustainable Reinforcement Learning

Srivatsan Krishnan[*] **Harvard University**
Maximilian Lam[*] **Harvard University**
Sharad Chitlangia[*†] **BITS Pilani Goa**
Zishen Wan[‡] **Harvard University**
Gaberial Barth-Maron **DeepMind**
Aleksandra Faust **Google Brain Research**
Vijay Janapa Reddi **Harvard University**

Abstract

Deep reinforcement learning continues to show tremendous potential in achieving task-level autonomy, however, its computational and energy demands remain prohibitively high. In this paper, we tackle this problem by applying quantization to reinforcement learning. To that end, we introduce a novel Reinforcement Learning (RL) training paradigm, *ActorQ*, to speed up actor-learner distributed RL training. *ActorQ* leverages 8-bit quantized actors to speed up data collection without affecting learning convergence. Our quantized distributed RL training system, *ActorQ*, demonstrates end-to-end speedups of $> 1.5\times$ - $2.5\times$, and faster convergence over full precision training on a range of tasks (Deepmind Control Suite) and different RL algorithms (D4PG, DQN). Furthermore, we compare the carbon emissions (Kgs of CO2) of *ActorQ* versus standard reinforcement learning on various tasks. Across various settings, we show that *ActorQ* enables more environmentally friendly reinforcement learning by achieving $2.8\times$ less carbon emission and energy compared to training RL-agents in full-precision. Finally, we demonstrate empirically that aggressively quantized RL-policies (up to 4/5 bits) enable significant speedups on quantization-friendly (supports native quantization) resource-constrained edge devices, without degrading accuracy. We believe that this is the first of many future works on enabling computationally /energy efficient and sustainable reinforcement learning. The source code for QuaRL is available here: `https://github.com/harvard-edge/QuaRL`.

1. Introduction

Deep reinforcement learning has attained significant achievements in various fields (Bellemare et al., 2012; Kempka et al., 2016; Kalashnikov et al., 2018; Silver et al., 2016, 2017; OpenAI, 2018; Chiang et al., 2019; OpenAI et al., 2019). Despite its promise, one of its limiting factors is long training times, and the current approach to speed up RL training involves distributed training (Espeholt et al., 2019a; Nair et al., 2015a; Babaeizadeh et al., 2016). Although distributed RL training has demonstrated significant potential in reducing training times, this approach also leads to increased energy consumption and greater carbon emissions

∗. Equal contribution.
†. This work was done while Sharad was a visiting student at Harvard.
‡. Now at Georgia Tech.

leading. To that end, we tackle the following research question – *How can we speed up RL training without significantly increasing its energy consumption and carbon emissions?*

To systematically tackle this problem, we first thoroughly characterize the performance of core components of distributed RL training. We find that majority of the time is spent on actor policy inference, followed by the learner's gradient calculation, model update, and finally the communication cost between actors and learners (Fig. 2). Thus, to obtain significant speedups, we first need to lower the the overhead of performing actor inference. To achieve this goal, we employ neural network quantization, a simple yet effective optimization technique to lower the compute and memory costs of neural network inference. Despite significant research on quantization for neural networks, to the best of our knowledge, there exists little prior work on applying quantization to speed up distributed reinforcement learning.

Applying quantization to reinforcement learning is non-trivial and different from traditional neural network quantization. In the context of policy inference, it may seem that, due to the sequential decision-making nature of reinforcement learning, errors made at one state might propagate to subsequent states, suggesting that policies might be more challenging to quantize than traditional neural network applications. In the context of reinforcement learning training, quantization seems difficult to apply due to the myriad of different learning algorithms (Lillicrap et al., 2015; Mnih et al., 2016; Barth-Maron et al., 2018) and the complexity of these optimization procedures. On the former point, our insight is that reinforcement learning policies are resilient to quantization error as policies are often trained with noise (Igl et al., 2019; Plappert et al., 2017) for exploration, making them robust. And on the latter point, we leverage the fact that reinforcement learning procedures may be framed through the actor-learner training paradigm (Horgan et al., 2018), and rather than quantizing learner optimization, we may achieve speedups while maintaining convergence by quantizing just the actors' experience generation. Through these insights, we successfully quantize deep reinforcement learning policies to speed up training time, reduce deployment costs and minimize carbon emissions.

In summary, our fundamental contributions are as follows:

- We introduce *ActorQ*, to speed up distributed reinforcement learning training. *ActorQ* operates by quantizing the actor's policy, thereby speeding up experience collection. *ActorQ* achieves between 1.5× and 2.5× speedup on a variety of tasks from the Deepmind control suite (Tassa et al., 2018) compared to its full-precision counterparts.

- Using our *ActorQ* framework, we further explore opportunities to identify various bottlenecks in distributed RL training. We show that quantization can also minimize communication overheads between actors and learners and reduce reinforcement learning time.

- Finally, by quantizing the policy weights and communication between actors and learners, we show a reduction in *carbon emissions* by as much as 2.8 × versus full precision policies, thus paving way for sustainable reinforcement learning.

- To address lack of benchmarks on quantization for reinforcement learning in the literature, we extensively benchmark quantized policies on standard tasks (Atari, Gym), algorithms (A2C, DDPG, DQN, D4PG, PPO), and models (MLPs, CNNs).

Metrics	A3C	CULE	Ray	Gorila	Seed-RL	ACME	Actor-Q (Our Work)
Method	Distributed	Distributed	Distributed	Distributed	Distributed	Distributed	Distributed/Standalone
Increase in Speed-up	✓	✓	✓	✓	✓	✓	✓
Decrease in Carbon Emission	×	×	×	×	×	×	✓
Decrease in Energy	×	×	×	×	×	×	✓
Decrease in Communication Cost	×	×	×	×	×	×	✓
Framework Agnostic	✓	✓	×	×	×	×	✓

Table 1: Comparison of prior works on speeding-up RL training wrt to speed-up (lower training times), energy, and carbon emissions. Previous works compared include Nvidia's CULE (Dalton et al., 2019), Ray (Moritz et al., 2017), Gorila (Nair et al., 2015c), Seed-RL (Espeholt et al., 2019b) and ACME (Hoffman et al., 2020a)

We demonstrate little to no loss in reward, especially in the context of post training quantization.

2. Related Work

Both quantization and reinforcement learning in isolation have been the subject of much research in recent years. Below we provide an overview of related works in both quantization and reinforcement learning and discuss their contributions and significance in relation to our paper.

2.1 Quantization

Quantizing a neural network reduces the precision of neural network weights, reducing memory transfer times and enabling the use of fast low-precision compute operations. Innovations in both post-training quantization (Krishnamoorthi, 2018a; Banner et al., 2018; Zhao et al., 2019; Tambe et al., 2019) and quantization aware training (Dong et al., 2019; Hubara et al., 2018; Choi et al., 2018) demonstrated that neural networks may be quantized to very low precision without accuracy loss, suggesting that quantization has immense potential for producing efficient deployable models. In the context of speeding up training, research has also shown that quantization can yield significant performance boosts. For example, prior work on half or mixed precision training (Sun et al., 2019; Das et al., 2018) demonstrates that using half-precision operators may significantly reduce compute and memory requirements while still achieving adequate convergence.

While much research has been conducted on quantization and machine learning, the primary targets of quantization are applications in the image classification and natural language processing domains. Quantization as applied to reinforcement learning has been relatively absent in the literature.

2.2 Reinforcement Learning & Distributed Reinforcement Learning Training

Significant work on reinforcement learning range from training algorithms (Mnih et al., 2013b; Levine et al., 2015) to environments (Brockman et al., 2016a; Bellemare et al., 2013; Tassa et al., 2018) to systems improvements (Petrenko et al., 2020; Hoffman et al., 2020b). From

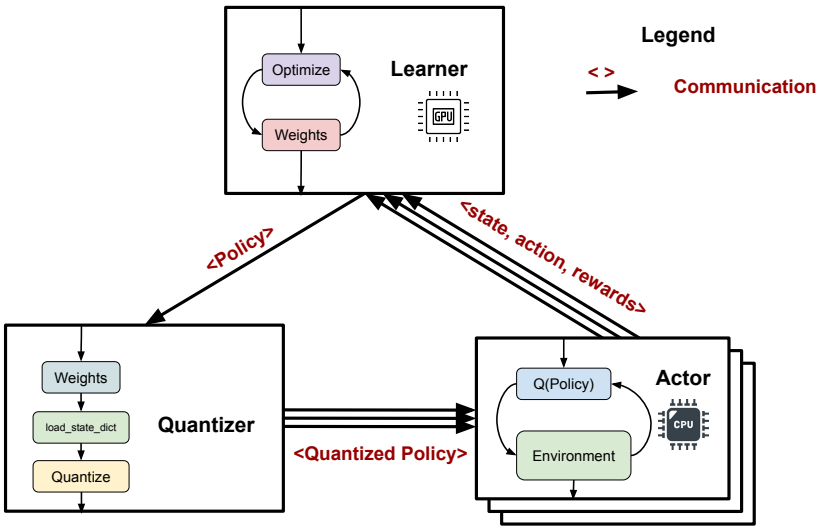


Figure 1: *ActorQ* system setup. *ActorQ* performs full precision GPU computation on the learner process and quantized CPU execution on the actor processes. We introduce a parameter quantizer to facillitate quantized communication of the updated policy between the GPU learner and CPU actors. The leaner and actors are instrumented with carbon monitoring APIs (Henderson et al., 2020) to quantify the impact carbon emission with and without quantization.

a systems angle, reinforcement learning poses a unique opportunity (as opposed to standard machine learning methods) as training a policy involves executing policies on environments (experience generation) and optimization (learning). Experience generation is trivially parallelizable and various recent research in distributed and parallel reinforcement learning training (Kapturowski et al., 2018; Moritz et al., 2018; Nair et al., 2015b) leverage this to accelerate training. One significant work is the Deepmind Acme reinforcement learning framework (Hoffman et al., 2020b), which enables scalable training to many processors or nodes on a single machine.

3. ActorQ: Quantization for Reinforcement Learning

We first provide a high-level overview of the *ActorQ* system. Next, we characterize the effects of quantization on different reinforcement learning algorithms. Lastly,we apply quantization to a distributed RL training framework to show speed-ups on a real system. Our results demonstrate that apart from reducing training time, *ActorQ* also leads to lower carbon emissions, thus paving the path for sustainable reinforcement learning research.

3.1 *ActorQ* System Architecture

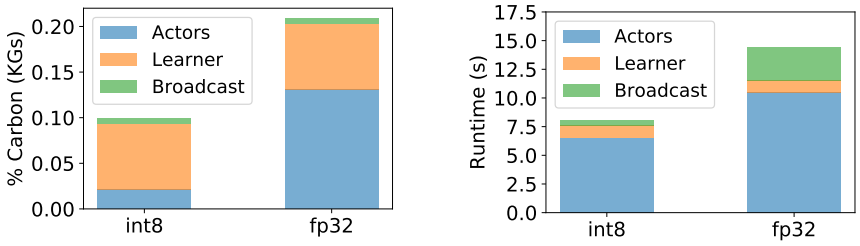
We introduce *ActorQ* for quantized actor-learner training. *ActorQ* utilizes quantization in the actor-learner reinforcement learning framework to speed up training. There are three main components in *ActorQ* system namely, 'Actors', 'Learner', and 'Quantizer' as

shown in Fig. 1. During training, each actor instance performs rollouts and initially uses a randomly initialized policy for decision making. At each step, the actors broadcast the environment state, action, and the reward for a given state to the learner. The learner uses this information to optimize the policy. Periodically, the learner broadcasts the updated policy to the actors, who then uses the updated policy to perform future rollouts.

There are two main performance bottlenecks in reinforcement learning training. First, each actor uses a neural network policy to generate an action. Thus, how fast it can perform rollouts depends on the policy's inference latency. Second, the learner broadcasts the policy periodically to all the actors. Broadcasting of the entire policy network to all actors can cause communication overheads and slow down training.

In ActorQ, we use quantization to reduce these bottlenecks and achieve end-to-end speed-up. To systematically integrate quantization into the training algorithm, we first perform a study to characterize the effects of applying post-training quantization and quantization aware training to various RL algorithms and environments.

In *ActorQ*, all the actor uses quantized policy to perform rollouts. Additionally, the broadcasted policy is also quantized and replaces the actor's policy. Note that *ActorQ* maintains all learner computation in full precision as to maintain learning convergence; further note that the learner is significantly faster than the actors due to utilizing the GPU (with actor policy inference achieving poor utilization on GPU due to operating over low batch sizes). Based on our characterization, we observe that time spent by actors to perform rollouts is significantly greater than that of learners. This motivates us to first apply quantization to the actors and policy broadcast communication from learners to actors.



(a) Carbon breakdown

(b) Time breakdown

Figure 2: Breakdown of carbon emissions for quantized and non-quantized training in MountainCar task.

While simple, *ActorQ* distinguishes from traditional quantized neural network training as the inference-only role of actors enables the use of low precision ($\leq$ 8 bit) operators to speed up training. This is unlike traditional quantized neural network training, which must utilize more complex algorithms like loss scaling Das et al. (2018), specialized numerical representations Sun et al. (2019); Wang et al. (2018), stochastic rounding Wang et al. (2018) to attain convergence. This adds extra complexity and may also limit speedup and, in many cases, are still limited to half-precision operations due to convergence issues.

3.2 Effects of Quantization on Reinforcement Learning

Before implementing *ActorQ*, we first perform experiments to understand the effects of quantization on reinforcement learning. Insights gained from these experiments will help verify that quantization can be applied to learning without significantly degrading quality.

To this end, we perform two types of quantization: post-training quantization (PTQ) and quantization aware training (QAT). The goal is to understand two things: First, does the quantization error at each step causes drift due to the feedback nature of RL. Second, how far can we quantize the RL policies (i.e., number of bits) before observing significant loss in rewards?

We take an RL policy fully trained in fp32 and apply post-training quantization to do the former. For the latter, i.e., to determine how far we can quantize the policy, we perform quantization aware training to simulate quantization effects using a straight-through estimator. The learning from these studies guides us in applying quantization for the *ActorQ* system to speed up the RL training.

Post-Training Quantization

The post-training quantization is performed using standard uniform affine quantization defined as follows:

$$Q_n(W) = round(\frac{W}{\delta})$$

where

$$\delta = \frac{|min(W, 0)| + |max(W, 0)|}{2^n}$$

Dequantization is defined as

$$D(W_q, \delta) = \delta(W_q)$$

In our study, policies are trained in standard full precision. Once trained, they are evaluated while applying the quantization (fp16 and int8) and dequantization functions to weights and activations to simulate quantization error.

For convolutional neural networks, we use per-channel quantization, which applies Q_n to each channel of convolutions individually. Also, all layers of the policy are quantized to the same precision level.

We apply the PTQ to Atari arcade learning (Bellemare et al., 2012), OpenAI gym environments (Brockman et al., 2016b) and different RL algorithms namely A2C (Mnih et al., 2016), DQN (Mnih et al., 2013a), PPO (Schulman et al., 2017), and DDPG (Lillicrap et al., 2015) and. We train a three-layer convolutional neural network for all Atari games for 10 million steps, with a quant delay of 5M (quantization aware training starts at 5M steps). For Gym environments, we train neural networks with two hidden layers of size 64. In PTQ, unless otherwise noted, both weights and activations are quantized to the same precision.

Table 2 shows the rewards attained by policies quantized via post-training quantization in. The mean of 8-bit and 16-bit relative errors ranges between 2% and 5% of the full precision model, which indicates that models may be quantized to 8/16 bit precision without much quality loss.

The overall performance difference between the 8-bit and 16-bit post-training quantization is minimal (with the exception of the DQN algorithm). Based on our analysis (See appendix), we believe this is because the policy's weight distribution is narrow enough that 8 bits can capture the distribution of weights without much error.

Algorithm →	**A2C**			**DQN**			**PPO**			**DDPG**		
Datatype →	**fp32**	**fp16**	**int8**	**fp32**	**fp16**	**int8**	**fp32**	**fp16**	**int8**	**fp32**	**fp16**	**int8**
Environment ↓	**Rwd**	**Rwd**	**Rwd**	**Rwd**	**Rwd**	**Rwd**	**Rwd**	**Rwd**	**Rwd**	**Rwd**	**Rwd**	**Rwd**
Breakout	379	371	350	214	217	78	400	400	368			
SpaceInvaders	717	667	634	586	625	509	698	662	684			
BeamRider	3087	3060	2793	925	823	721	1655	1820	1697			
MsPacman	1915	1915	2045	1433	1429	2024	1735	1735	1845			
Qbert	5002	5002	5611	641	641	616	15010	15010	14425			
Seaquest	782	756	753	1709	1885	1582	1782	1784	1795			
CartPole	500	500	500	500	500	500	500	500	500			
Pong	20	20	19	21	21	21	20	20	20			
Walker2D	399	422	442				2274	2273	2268	1890	1929	1866
HalfCheetah	2199	2215	2208				3026	3062	3080	2553	2551	2473
BipedalWalker	230	240	226				304	280	291	98	90	83
MountainCar	94	94	94				92	92	92	92	92	92

Table 2: Post-training quantization error for DQN, DDPG, PPO, and A2C algorithm on Atari and Gym. Quantization down to 8 bits yields similar rewards to full precision baseline.

In a few cases (e.g., MsPacman for PPO), post-training quantization yields better scores than the full precision policy. We believe that quantization injected an amount of noise that was small enough to maintain a good policy and large enough to regularize model behavior; this supports some of the results seen by Louizos et al. (2018); Bishop (1995); Hirose et al. (2018).

In summary, based on this study, we observe that quantization of RL policy does not cause significant loss in reward compared to an fp32 policy.

Also, it is important to note that we use simple uniform affine quantization to demonstrate how to apply quantization in RL policies; however, we can easily swap the quantizer function to include other quantization techniques Krishnamoorthi (2018b).

Quantization Aware Training

To understand how aggressively (i.e., number of bits) we can quantify the RL policies, we use quantization aware training (QAT). In QAT, the RL policy weights and activations are passed through the quantization function Q_n during inference; during backpropagation the straight-through estimator is used as the gradient of Q_n

$$\nabla_W Q_n(W) = I$$

Note that quantization aware training does not speed up training as all operations are still in floating point. Quantization aware training is used primarily to train a model with quantized weights and activations to evaluate the reward loss (if any) for a given RL task. Quantifying this allows us to speed policy inference time with quantized execution.

We present rewards for policies quantized via quantization aware training on multiple environments and training algorithms in Figure 3. Generally, the performance relative to

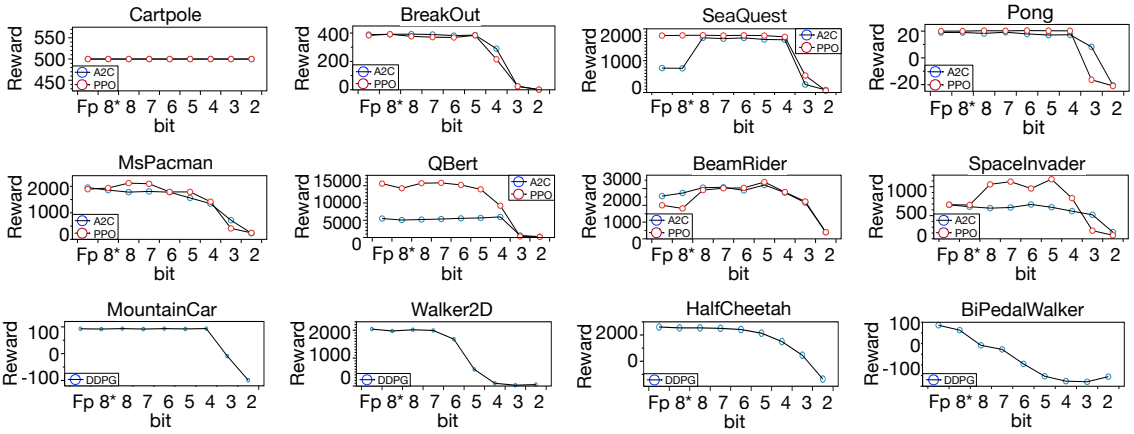


Figure 3: Quantization aware training of PPO, A2C, and DDPG algorithms on OpenAI gym, Atari, and PyBullet. FP is achieved by fp32 and 8* is achieved by 8-bit post-training quantization.

the full precision baseline is maintained until 5/6-bit quantization, after which there is a drop in reward.

Broadly, at 8-bits, we see no degradation in rewards. Hence, when applying quantization in *ActorQ*, we quantize the policy to 8-bits. By quantizing the policy at 8-bits and leveraging the native 8-bit computation support in hardware, we achieve end-to-end speed-up in reinforcement learning training.

4. Results

We evaluate the *ActorQ* system for speeding up distributed quantized reinforcement learning across various tasks in Deepmind Control Suite (Tassa et al., 2018). Overall, we show that: (1) we see significant speedup (>1.5 ×-2.5 ×) in training reinforcement learning policies using *ActorQ*; (2) convergence is maintained even when actors perform down to 8-bit quantized inference; (3) Using *ActorQ*, we lower the carbon emissions from 48% to 73% compared to training without quantization.

4.1 *ActorQ Experimental Setup*

We evaluate *ActorQ* on a range of environments from the Deepmind Control Suite (Tassa et al., 2018). We choose the environments to cover a wide range of difficulties to determine the effects of quantization on both easy and difficult tasks. The difficulty of the Deepmind Control Suite tasks is determined by (Hoffman et al., 2020b). Table 3 lists the environments we tested on with their corresponding difficulty and number of steps trained. Each episode has a maximum length of 1000 steps, so the maximum reward for each task is 1000 (though this may not always be attainable).

Task	Algorithm	Difficulty	Steps Trained	Model Pull Freq (steps)
Cartpole Balance	D4PG	Trivial	40000	1000
Walker Stand	D4PG	Trivial	40000	1000
Hopper Stand	D4PG	Easy	100000	1000
Reacher Hard	D4PG	Easy	70000	1000
Cheetah Run	D4PG	Medium	200000	1000
Finger Spin	D4PG	Medium	200000	1000
Humanoid Stand	D4PG	Hard	500000	100
Humanoid Walk	D4PG	Hard	700000	100
Cartpole	DQN	N/A	60000	1000
Acrobot	DQN	N/A	100000	1000
MountainCar	DQN	N/A	200000	1000

Table 3: Tasks evaluated using *ActorQ* range from easy to difficult, along with the steps trained for corresponding tasks, with how frequently the model is pulled on the actor side.

Policy architectures are fully connected networks with three hidden layers of size 2048. We apply a gaussian noise layer to the output of the policy network on the actor to encourage exploration; sigma is uniformly assigned between 0 and 0.2 according to the actor being executed. On the learner side, the critic network is a three-layer hidden network with a hidden size of 512. We train policies using D4PG (Barth-Maron et al., 2018) on continuous control environments and DQN (Mnih et al., 2013b) on discrete control environments. We chose D4PG as it was the best learning algorithm in (Tassa et al., 2018; Hoffman et al., 2020b), and DQN is a widely used and standard reinforcement learning algorithm. An example submitted by an actor is sampled 16 times before being removed from the replay buffer (spi=16) (lower spi is typically better as it minimizes model staleness (Fedus et al., 2020)).

All the experiments are run in a distributed fashion to leverage multiple CPU cores and a GPU. A V100 GPU is used on the learner, while the actors are mapped to the CPU (1 core for each actor). We run each experiment and average over at least three runs to compute the running mean (window=10) of the aggregated runs.

Measuring Carbon Emissions. For measuring the carbon emission for the run, we use the `experiment-impact-tracker` proposed in prior JMLR work (Henderson et al., 2020)[1]. We instrument the *ActorQ* system with carbon monitor APIs to measure the energy and carbon emissions for each training experiment in *ActorQ*.

4.2 Experimental Results

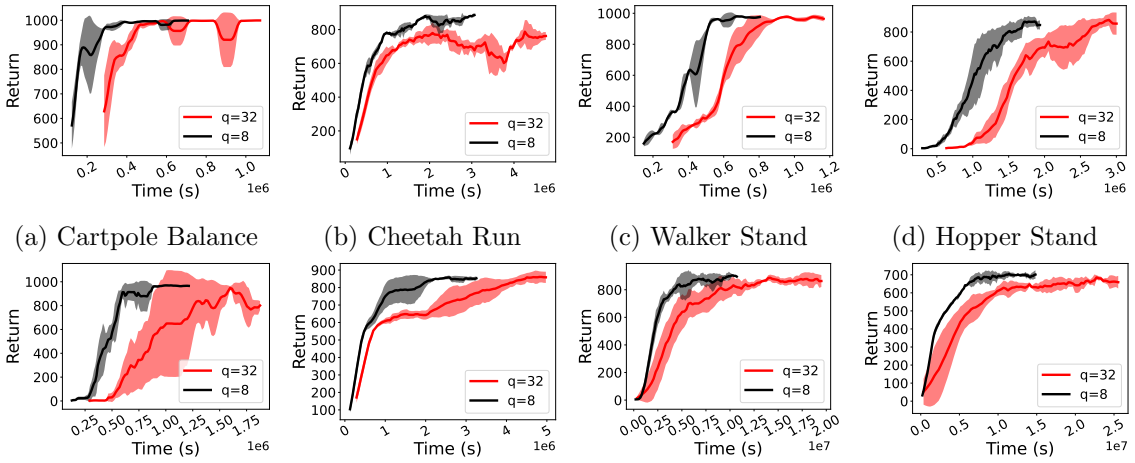
End to End Speedups. We show end to end training speedups with *ActorQ* in Figure(s) 4 and 6. Across nearly all tasks, we see significant speedups with both 8-bit inference. Additionally, to improve readability, we estimate the 95% percentile of the maximum attained score by fp32 and measure time to this reward level for fp32, int8, and compute corresponding speedups. This is shown in Table 4. Note that Table 4 does not take into account cases where fp16 or int8 achieve a higher score than fp32.

Convergence. We show the episode reward versus total actor steps convergence plots using *ActorQ* in Figure(s) 5 and 6. Data shows that broadly, convergence is maintained

1. `https://github.com/Breakend/experiment-impact-tracker`

Task	Reward Achieved	Time to Reward (s)		Speedup	Carbon (kg)		Carbon Reduction
		fp32	int8	int8	fp32	int8	int8
Cartpole Balance	941.22	870.91	279.00	3.12×	0.359	0.15	58.28%
Walker Stand	947.74	871.32	534.37	1.63×	0.67	0.178	73.41%
Hopper Stand	836.41	2660.41	1699.17	1.57×	0.34	0.17	50.00%
Reacher Hard	948.12	1597.00	875.34	1.82×	0.35	0.18	48.57%
Cheetah Run	732.31	2517.30	891.84	2.82×	0.263	0.12	54.37%
Finger Spin	810.32	3256.56	1065.52	3.06×	0.361	0.19	47.37%
Humanoid Stand	884.89	13964.92	9302.82	1.51×	0.55	0.27	50.91%
Humanoid Walk	649.91	17990.66	6223.35	2.89×	0.56	0.278	50.36%
Cartpole (Gym)	198.22	963.67	260.10	3.70×	0.188	0.089	52.50%
Mountain Car (Gym)	-120.62	2861.80	1284.32	2.22×	0.21	0.098	53.27%
Acrobot (Gym)	-107.45	912.24	168.44	5.41×	0.198	0.097	50.97%

Table 4: *ActorQ* time and speedups to 95% reward on select tasks from Deepmind Control Suite and Gym. 8 bit inference yields $> 1.5\times - 2.5\times$ speedup over full precision training. We use D4PG on DeepMind Control Suite environments (non-gym), DQN on gym environments.



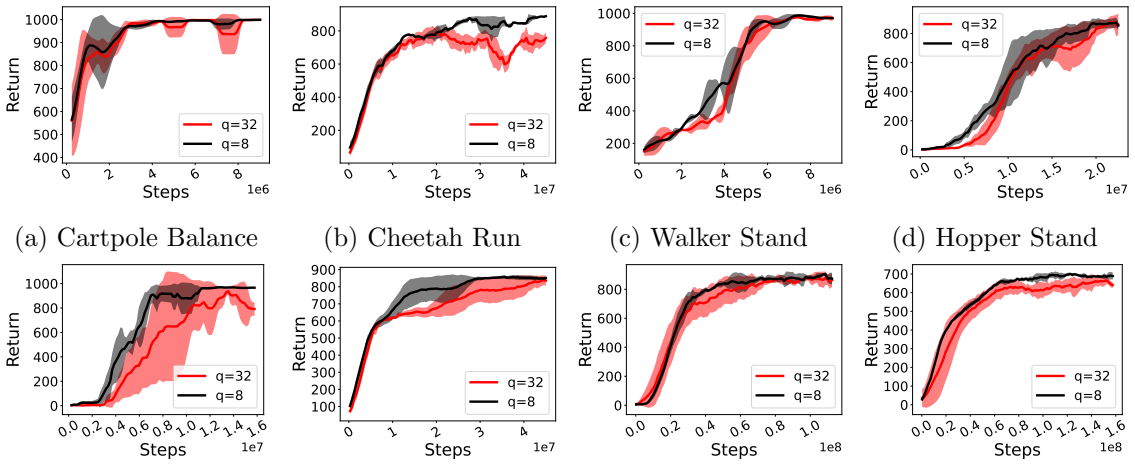
(a) Cartpole Balance (b) Cheetah Run (c) Walker Stand (d) Hopper Stand

(e) Reacher Hard (f) Finger Spin (g) Humanoid Stand (h) Humanoid Walk

Figure 4: End to end speedups of *ActorQ* across various Deepmind Control Suite tasks using 8 bit and 32 bit inference. 8 bit training yields significant end to end training speedups over the full precision baseline. The x-axis denotes the wall-clock time and y-axis denotes the reward. Training uses the D4PG algorithm.

even with 8-bit actors across both easy and difficult tasks. On Cheetah, Run and Reacher, Hard, 8-bit *ActorQ* achieve even slightly faster convergence, and we believe this may have happened as quantization introduces noise which could be seen as exploration.

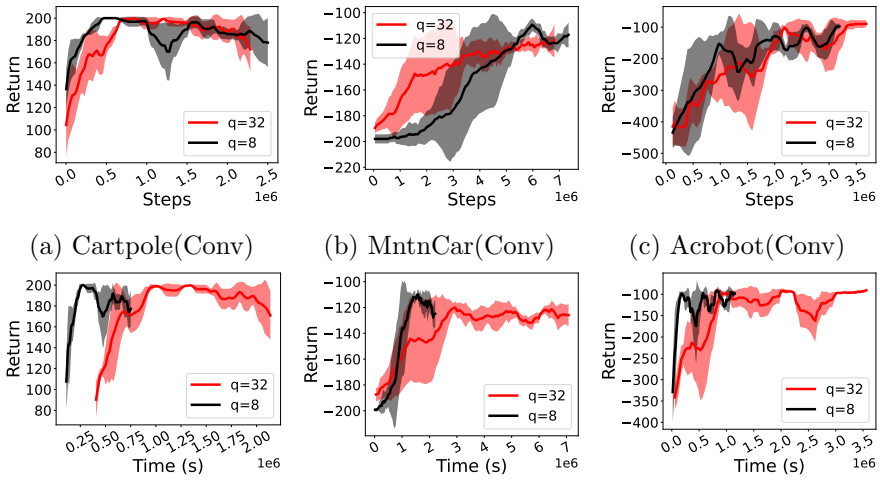
Carbon Emissions. Table 4 also shows the carbon emissions for various task in openAI gym and Deepmind Control Suite. We compare the carbon emissions of a policy running in fp32 and int8. We observe that quantization of policies reduces the carbon emissions anywhere from 48% to 73.41% depending upon the task. As RL systems are scaled to run on 1000's distributed CPU cores and accelerators (GPU/TPU), the carbon reduction can be significant.



(a) Cartpole Balance (b) Cheetah Run (c) Walker Stand (d) Hopper Stand

(e) Reacher Hard (f) Finger Spin (g) Humanoid Stand (h) Humanoid Walk

Figure 5: Convergence of *ActorQ* across various Deepmind Control Suite tasks using 8 bit and 32 bit inference. 8 bit quantized training attains the same or better convergence than full precision training. Training uses the D4PG algorithm.



(a) Cartpole(Conv) (b) MntnCar(Conv) (c) Acrobot(Conv)

(d) Cartpole(Spd) (e) MntnCar(Spd) (f) Acrobot(Spd)

Figure 6: Convergence (Conv.) and end to end speedups (Spd) of *ActorQ* across various Gym tasks using 8 bit and 32 bit inference. 8 bit training yields significant end to end training speedups over the full precision baseline. For the speed-up plots (d-f), the x-axis denotes wall-clock time. Training uses the DQN algorithm.

4.3 Communication vs Computation

The frequency of model pulls on actors is a hyperparameter and may have impacts on convergence as it affects the staleness of policies being used to populate the replay buffer; this has been witnessed in both prior research (Fedus et al., 2020) and our experiment with the hyperparameter. Figure 7 shows that a higher update frequency of 100 can help in faster convergence compared to an update frequency of 1000 for the Humanoid stand task.

This hyperparameter has system-level implications since a higher update frequency can increase the communication cost (policy broadcast from learner to actors). In contrast, a lower update frequency can increase the computation cost since it will take more steps to converge, increasing the computation cost. Thus, to understand the tradeoff of quantization concerning this hyperparameter, we explore the effects of quantization of communication versus computation in both communication and computation-heavy setups.

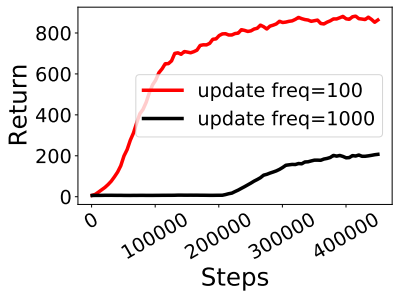


Figure 7: Humanoid stand: training with more frequent actor pulls learns faster than with less frequent actor pulls and demonstrates model pull frequency affects staleness of actor policies and may have an effect on training.

To quantize communication, we quantize policy weights to 8 bits and compress them by packing them into a matrix, thus, reducing the memory of model broadcasts by 4×. Naturally, quantizing communication would be more beneficial in the communication heavy scenario, and quantizing compute would yield relatively more gains in the computation-heavy scenario.

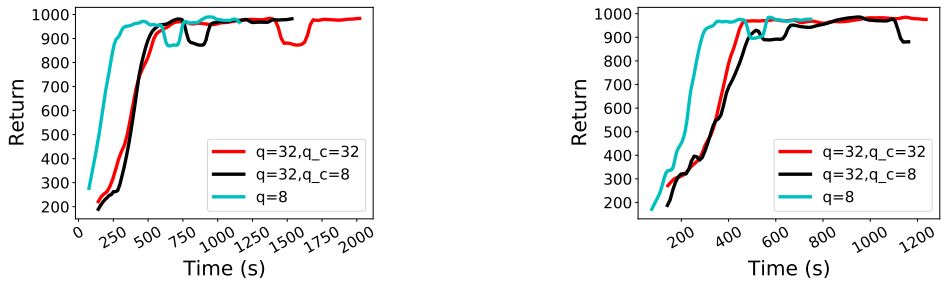
Figure 8 shows an ablation plot of the gains of quantization on both communication and computation in a communication heavy scenario (frequency=30) versus a computation-heavy scenario (frequency=1000).

Figures show that in a communication heavy scenario, quantizing communication may yield up to 30% speedup; conversely, in a computation-heavy scenario quantizing communication has little impact as the overhead is dominated by computation. Therefore, we believe that communication would incur higher costs on a networked cluster as actors scale.

4.4 Why Quantization of Actors Speed-Up RL Training

We further break down the various components contributing to runtime on a single actor to understand how quantization of actor's policy inference speeds up training. Runtime components are broken down into: step time, pull time, deserialize time, and load_state_dict time. Step time is the time spent performing neural network policy inference. Pull time is the time between querying the Reverb queue (DeepMind, 2020) for a model and receiving the serialized models' weights; deserialize time is the time spent to deserialize the serialized model dictionary; load_state_dict time is the time to call PyTorch load_state_dict (used for loading and storing the policy).

Figure 9a shows the relative breakdown of the component runtimes with 32, 16, and 8-bit quantized inference in the computation heavy scenario. As shown, step time is the main bottleneck, and quantization of the actor's policy significantly speed-up each roll-out,

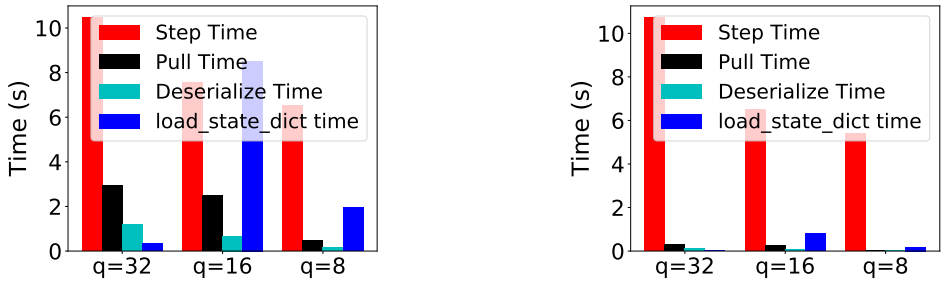


(a) Communication Heavy (Update Freq=30) (b) Computation Heavy (Update Freq=300)

Figure 8: Effects of quantizing communication versus computation in compute heavy and communication heavy training scenarios. q is the precision of inference; q_c is the precision of communication. Note q=8 implicitly quantizes communication to 8 bits. Experiment run on the walker stand task, using the D4PG algorithm.

speeding up the overall training. Figure 9b shows the cost breakdown in the communication heavy scenario. While speeding up computation, pull time and deserialize time are also significantly sped up by quantization due to reduction in memory.

In 8-bit and 16-bit quantized training, the cost of PyTorch load_state_dict is significantly higher. An investigation shows that the cost of loading a quantized PyTorch model is spent repacking the weights from Python object into C data. 8-bit weight repacking is noticeably faster than 16-bit weight repacking due to fewer memory accesses. The cost of model loading suggests that additional speed gains can be achieved by serializing the packed C data structure and reducing the cost of weight packing.



(a) Communication Heavy (Update Freq=30) (b) Computation Heavy (Update Freq=300)

Figure 9: Breakdown of runtime components for quantized and non-quantized training over 1000 steps.

5. Discussion

We demonstrate that quantization for reinforcement is a simple method to achieve sustainable reinforcement learning. *ActorQ* is an example of how quantization can be applied to distributed reinforcement learning to achieve lower training times and carbon emissions. As we scale RL training to a hundred thousand cores and GPUs, we believe even 50% improvement will result in enormous savings in dollar cost, energy, and carbon emissions.

However, there are some aspects of applying quantization to reinforcement learning that warrants further discussion and research.

In our design of quantizer in *ActorQ*, we relied on simple uniform quantization. However, we believe other forms of aggressive quantization can also be applied. However, two critical considerations need to be taken into account to achieve actual system level speedup. First, the quantization method should have little overheads from a system level. Second, the hardware must natively support those quantized ops (e.g., int8 instruction support).

Also, in *ActorQ*, we primarily focused on quantizing actors and communication of policy from learners to the actors since these two were the biggest computation bottlenecks in training time and carbon emissions. However, we believe that the learner's policy can also be quantized.

6. Conclusion

We evaluate quantization to speed up reinforcement learning training and inference. We show standard quantization methods can quantize policies down to ≤ 8 bits with little loss in quality. We develop *ActorQ*, and attain significant speedups over full precision training. Our results demonstrate that quantization has considerable potential in speeding up both reinforcement learning inference and training. Future work includes extending the results to networked clusters to evaluate further the impacts of communication and applying quantization to reinforcement learning to different application scenarios such as the edge.

References

Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment, 2018.

Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. 2018.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, jun 2013.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL `http://arxiv.org/abs/1207.4708`.

C. M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, Jan 1995. doi: 10.1162/neco.1995.7.1.108.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016a. URL `http://arxiv.org/abs/1606.01540`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016b. URL `http://arxiv.org/abs/1606.01540`.

Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4 (2):2007–2014, April 2019. ISSN 2377-3766. doi: 10.1109/LRA.2019.2899918.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. 2018.

Steven Dalton, Iuri Frosio, and Michael Garland. Gpu-accelerated atari emulation for reinforcement learning, 2019.

Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, Alexander Heinecke, Pradeep Dubey, Jesus Corbal, Nikita Shustrov, Roma Dubtsov, Evarist Fomenko, and Vadim Pirogov. Mixed precision training of convolutional neural networks using integer operations. *International Conference on Learning Representations (ICLR)*, 2018.

DeepMind. Reverb. `https://github.com/deepmind/reverb`, 2020.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 293–302, 2019.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. 2019a.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. *arXiv preprint arXiv:1910.06591*, 2019b.

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning (ICML)*, 2020.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43, 2020. URL `http://jmlr.org/papers/v21/20-312.html`.

Kazutoshi Hirose, Ryota Uematsu, Kota Ando, Kodai Ueyoshi, Masayuki Ikebe, Tetsuya Asai, Masato Motomura, and Shinya Takamaeda-Yamazaki. Quantization error-based regularization for hardware-aware neural network training. *Nonlinear Theory and Its Applications, IEICE*, 9(4):453–465, 2018. doi: 10.1587/nolta.9.453.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020a.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020b.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2018.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2018. URL `http://jmlr.org/papers/v18/16-456.html`.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, pages 13978–13990, 2019.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018. URL `http://arxiv.org/abs/1806.10293`.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018a.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018b.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies, 2015.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. *International Conference on Learning Representations (ICLR)*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013a.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013b.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL `http://arxiv.org/abs/1602.01783`.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications. *arXiv preprint arXiv:1712.05889*, 2017.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015a.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015b.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015c.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand, 2019.

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 *International Conference on Machine Learning (ICML)*, 2020.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *CoRR*, abs/1706.01905, 2017. URL http://arxiv.org/abs/1706.01905.

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

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi (Viji) Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. In *Advances in Neural Information Processing Systems 32*. 2019.

Thierry Tambe, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, David Brooks, and Gu-Yeon Wei. Adaptivfloat: A floating-point based data type for resilient deep learning inference. *arXiv preprint arXiv:1909.13271*, 2019.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Advances in Neural Information Processing Systems*, 2018.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. *International Conference on Machine Learning (ICML)*, pages 7543–7552, June 2019.