Learning Quantized Continuous Controllers for Integer Hardware

Fabian Kresse FABIAN.KRESSE@IST.AC.AT
Christoph H. Lampert CHL@IST.AC.AT
Institute of Science and Technology Austria (ISTA), Am Campus 1, 3400 Klosterneuburg, Austria

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

Deploying continuous-control reinforcement learning policies on embedded hardware requires meeting tight latency and power budgets. Small FPGAs can deliver these, but only if costly floating-point pipelines are avoided. We study quantization-aware training (QAT) of policies for integer inference and we present a learning-to-hardware pipeline that automatically selects low-bit policies and synthesizes them to an Artix-7 FPGA. Across five MuJoCo tasks, we obtain policy networks that are competitive with full precision (FP32) policies but require as few as 3 or even only 2 bits per weight, and per internal activation value, as long as input precision is chosen carefully. On the target hardware, the selected policies achieve inference latencies on the order of microseconds and consume microjoules per action, favorably comparing to a quantized reference. Last, we observe that the quantized policies exhibit increased input noise robustness compared to the floating-point baseline.

Keywords: Quantization-aware training, reinforcement learning, FPGA, low-bitwidth quantization

1. Introduction

Deep Reinforcement Learning (RL) has achieved strong results in continuous control with real-time requirements, such as robotic manipulation (Kalashnikov et al., 2018), unmanned aerial vehicle (UAV) control (Kaufmann et al., 2023), and fusion plasma control (Degrave et al., 2022).

Beyond *high control quality*, practical deployment of such systems hinges on *latency*, *energy*, and *compatibility with the available hardware*: Constraints on *latency*, i.e. the sensor-to-actuator delay, usually determine which tasks are feasible: for autonomous-driving tasks, latency commonly has to stay below $100\,\mathrm{ms}$, even at modest driving speeds (Kato et al., 2015). For the inner loop of a robotic arm or quadcopter control, a latency of only a few milliseconds can be required (Zhang et al., 2020; Dimitrova et al., 2020). For special-purpose settings, such as nuclear fusion, controllers have to operate even faster, allowing for a latency of at most $100\,\mu\mathrm{s}$ per decision (Degrave et al., 2022). Note that inference is only one piece of the sense-act pipeline. Therefore, the actual compute budget available to the processing unit is even smaller. Furthermore, a low latency might also be required to shrink the sim-to-real gap: delayed sensing and actuation can degrade policy performance, requiring techniques to model or compensate for too-large latency (Tan et al., 2018; Schuitema et al., 2010).

One way to reduce latency is to move the policy to a faster compute platform, but this typically raises *energy consumption*. That trade-off is especially problematic on battery-powered systems, such as nano-/micro-UAVs, which operate under power envelopes of a few watts or less (Palossi et al., 2021), as reduced energy draw directly extends mission time before recharge. For comparison, high-end GPUs routinely draw hundreds of watts, e.g. up to $700\,\mathrm{W}$ for one NVIDIA H100 (NVIDIA, 2025).

Specialized embedded hardware, notably field-programmable gate arrays (FPGAs), are a promising way to achieve low latency at low power (Wan et al., 2021; Umuroglu et al., 2017). However, these gains come with constraints: limited on-chip memory and the high cost of floating-point (FP32) arithmetic push deployments toward low-precision formats (Rati et al., 2025; Umuroglu et al., 2017). Overall, there is no universal deployment recipe for neural network policies: latency, energy, memory footprint, and numerical precision requirements vary by task and platform. In this work, we concentrate on *continuous-control* neural network policies learned with RL in simulation, with the final inference target architecture being FPGAs.

As FP32 operations are costly in hardware, we restrict ourselves to *integer-only* operations, matching the computational abilities of FPGAs much better. Hence, standard FP32 networks cannot be deployed *as is*. A straightforward path is post-training quantization (PTQ), which converts a pretrained full-precision model to an integer-only one using calibration data and no (or minimal) retraining (Gholami et al., 2022). However, PTQ often underperforms approaches that train with quantization in the loop: *quantization-aware training* (QAT). Additionally, PTQ often performs worse on smaller models, which are investigated in this work (Jacob et al., 2018). Furthermore, Chandrinos et al. (2024) report QAT outperforming PTQ in multi-agent few-bit quantization settings.

Therefore, we adopt QAT for continuous-control policies, training networks directly in the quantized regime. QAT explicitly accounts for limited numerical precision and the non-differentiability of quantization during training (via straight-through estimators), yielding models trainable with gradient descent and compatible with integer-only deployment.

Contributions. We demonstrate that QAT can be successfully used for learning continuous-control policies, quantifying how far bitwidths and layer sizes can be reduced while preserving control quality relative to floating-point on MuJoCo tasks. We also examine the impact of quantization at the input interface. To probe robustness, we compare QAT and FP32 policies under input-noise injection. Finally, we synthesize trained policies to a representative FPGA using FINN (Blott et al., 2018), demonstrating better resource-latency trade-offs than a strong, already-quantized reference.

2. Quantized Reinforcement Learning for Integer Hardware

In this section, we outline our pipeline for training and deploying integer-only controllers: RL setup, QAT of the policy, and FPGA synthesis.

2.1. Deep reinforcement learning for continuous control

In reinforcement learning, an agent interacts with its environment and, through trial and error, learns to maximize cumulative reward (Sutton and Barto, 2018). This interaction is formalized as a Markov Decision Process (MDP) specified by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $P(s' \mid s, a)$ the transition dynamics, $R(s, a)$ the immediate reward, and $\gamma \in (0, 1]$ the discount factor. At each time-step t, the agent follows a policy π selecting an $a_t \in \mathcal{A}$ given the current state $s_t \in \mathcal{S}$; the environment then samples the next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$ and returns a reward $r_t = R(s_t, a_t)$. An *episode* is the resulting sequence $(s_0, a_0, r_0, s_1, \ldots)$, and the learning objective is to maximize the expected discounted return $G = \sum_{t=0}^{N-1} \gamma^t r_t$.

In this work, we consider the setting where both $\mathcal{S}$ and $\mathcal{A}$ are continuous and multi-dimensional. A variety of algorithms have been proposed to solve for maximizing the discounted return in this

setting. We focus on off-policy, actor-critic methods, specifically Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016) and Soft Actor-Critic (SAC) (Haarnoja et al., 2018). DDPG uses a deterministic *policy*, $\pi_\theta : \mathcal{S} \to \mathcal{A}$, that proposes actions for a given state, together with a *critic*, $Q_\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, that estimates expected returns. At training time, SAC employs a stochastic Gaussian policy $\pi_\theta(a \,|\, s) = \mathcal{N}\big(\mu_\theta(s), \mathrm{diag}(\sigma^2_\theta(s))\big)$ together with two critics. At deployment-time, SAC uses the deterministic maximum-likelihood action. In both cases, the parameters θ and ϕ are neural-network weights that are optimized end-to-end via gradient-based updates based on a replay buffer of transitions.

2.2. Quantization-aware training

We instantiate all neural networks, i.e. the policy and the critics, as standard full-precision (FP32) neural networks with ReLU non-linearities and a final hyperbolic tangent activation function that bounds action values to $[-1, 1]$, following the implementation in CleanRL (Huang et al., 2022). Because our goal is integer *inference*, and the critics networks can be discarded after training, we must only ensure that the policy network follows integer arithmetic. For SAC, the σ_θ branch is not needed at deployment; we therefore implement it as a separate FP32 subnetwork (one hidden layer with 64 units) used only during training.

To ensure the policy π_θ is compatible with integer arithmetic deployment, we impose *Quantize/De-Quantize* (QDQ) steps (Jacob et al., 2018), at the input, for all weights, after every ReLU, and before the final output. For a positive scale s and a FP32 input x, this is defined as

$$\mathrm{QDQ}_b(x; s) \;=\; \frac{s}{q_s}\,\mathrm{Q}_b(x; s), \qquad \mathrm{Q}_b(x; s) \;=\; \mathrm{clip}\left(\mathrm{round}\left(\tfrac{x}{s} q_s\right),\, q_{\min},\, q_{\max}\right), \tag{1}$$

where round rounds a value to its nearest integer. The integer bounds $(q_{\min}, q_{\max})$ are fixed by the bitwidth parameter b and the intended signedness of the operation: for the inputs, the network weights, and the final output we use *signed quantization*: $[q_{\min}, q_{\max}] = [-2^{b-1},\, 2^{b-1} - 1]$. For all intermediate layers, a ReLU activation precedes the QDQ step, so negative values cannot occur, and we use *unsigned quantization* to fully utilize the available bitwidth: $[q_{\min}, q_{\max}] = [0,\, 2^b - 1]$. The to-integer scaling factor q_s is defined as $q_s = \max(|q_{\min}|, |q_{\max}|)$. QDQ projects the FP32 value onto the integer lattice (via scaling, rounding, and clipping) and then de-quantizes it back to FP32, allowing us to retain a standard FP32 training loop while enforcing integer arithmetic at deployment.

Because rounding is piecewise constant and therefore non-differentiable, we use a *straight-through estimator* (STE) to backpropagate through QDQ: during the backward pass, we treat the quantizer as the identity for gradients (Bengio et al., 2013; Jacob et al., 2018).

In this work, we initialize the activation scales during a 300-step warm-up via an exponential moving high percentile of the input statistics and, after warm-up, we learned them by gradient updates. Weight scales are not learned: for each weight matrix, the scale is fixed to its absolute maximum.

2.3. Integer-only deployment

At deployment time, all quantization scales are fixed. For each weight tensor w with associated scale s_w, we straightforwardly obtain its integer representation $\tilde{w} = \mathrm{Q}_b(w; s_w)$. The input state x_0 is quantized on the fly using the floating-point input scale s_x, i.e., $\tilde{x}_0 = \mathrm{Q}_b(x_0; s_x)$, and thereafter

the network runs integer-only. Each layer performs integer matrix-vector products with sufficiently wide accumulators to avoid overflow; after accumulation ReLU is applied, and the result is *requantized* to the target bitwidth for the next layer, i.e., it is scaled according to the predetermined scale of the next layer and clipped to $[q_{\min}, q_{\max}]$. In our FINN-based deployment, this requantization is implemented based on stored thresholds (Blott et al., 2018), thus avoiding any FP32 operations. At the final layer, an analogous requantization takes place with a mapping to $[-1, 1]$ via a hyperbolic tangent (implemented as a lookup), yielding the action value for the environment.

Ultimately, the only floating-point operation involved is the initial quantization of the input state. We leave this unmodified, because for real-world systems the needed processing step depends on the sensors used. For example, sensor readings typically arrive as integers (e.g., from analog-digital converters), which can be scaled via integer arithmetic or lookup tables.

To deploy the quantized policy network on a hardware platform, it has to be *synthesized*. For this, we use the FINN dataflow compiler (Umuroglu et al., 2017; Blott et al., 2018), generating a *streaming dataflow*. Layers are connected by FIFO streams, and both weights and activations are kept on chip, eliminating external DRAM traffic and thereby reducing latency and energy consumption. Each layer exposes two parallelism variables: the number of *processing elements* (PEs; parallelism along the matrix rows / output features) and the *SIMD* width (parallel inputs consumed per cycle, i.e., columns). Increasing either reduces per-layer cycle count and raises throughput, at the cost of additional resources. FINN can also target a user-specified throughput and automatically choose PE and SIMD count to meet it.

3. Experiments

We evaluate QAT on five MuJoCo tasks (Humanoid-, Walker2d-, Ant-, HalfCheetah-, Hopper-v4) using SAC and DDPG. Our goal is to obtain *very small*, hardware-friendly policies that match FP32 returns. We structure the evaluation into three parts: (i) *bitwidth sensitivity*, where we vary which components of the policy are quantized to investigate where precision matters; (ii) *model selection*, a three-step rule that first fixes the smallest FP32 matching configuration for internal activations and weight bitwidth, then the hidden layer width, and last the input activation precision; and (iii) *robustness*, where we perform noise-injection tests on input states. Finally, we synthesize the selected policies to an Artix-7 FPGA with FINN v0.10.1, targeting a fixed 100 MHz clock and reporting latency, throughput, power usage, and resources utilized.

All training runs were performed with 1M environment steps; for evaluation, we average over 1,000 (10 where stated) deterministic policy rollouts per trained model, and report mean and standard deviation across 10 training seeds. All reported rewards are undiscounted.

Baseline (FP32) As a baseline, we train full-precision policies using the CleanRL (Huang et al., 2022) implementations of SAC and DDPG, utilizing corresponding default hyperparameters and model sizes (see Appendix A). In addition, we perform running normalization of the input state space (per-dimension, frozen at evaluation) for SAC, but not DDPG, as we find it to perform better.

Quantized Models We swap the policy network for a QAT variant implemented with Brevitas v0.12.0 (Franco et al., 2025), using the same hyperparameters as the FP32 baselines. In this case, we perform running normalization for both DDPG and SAC. An ablation on normalization can be found in Appendix C.

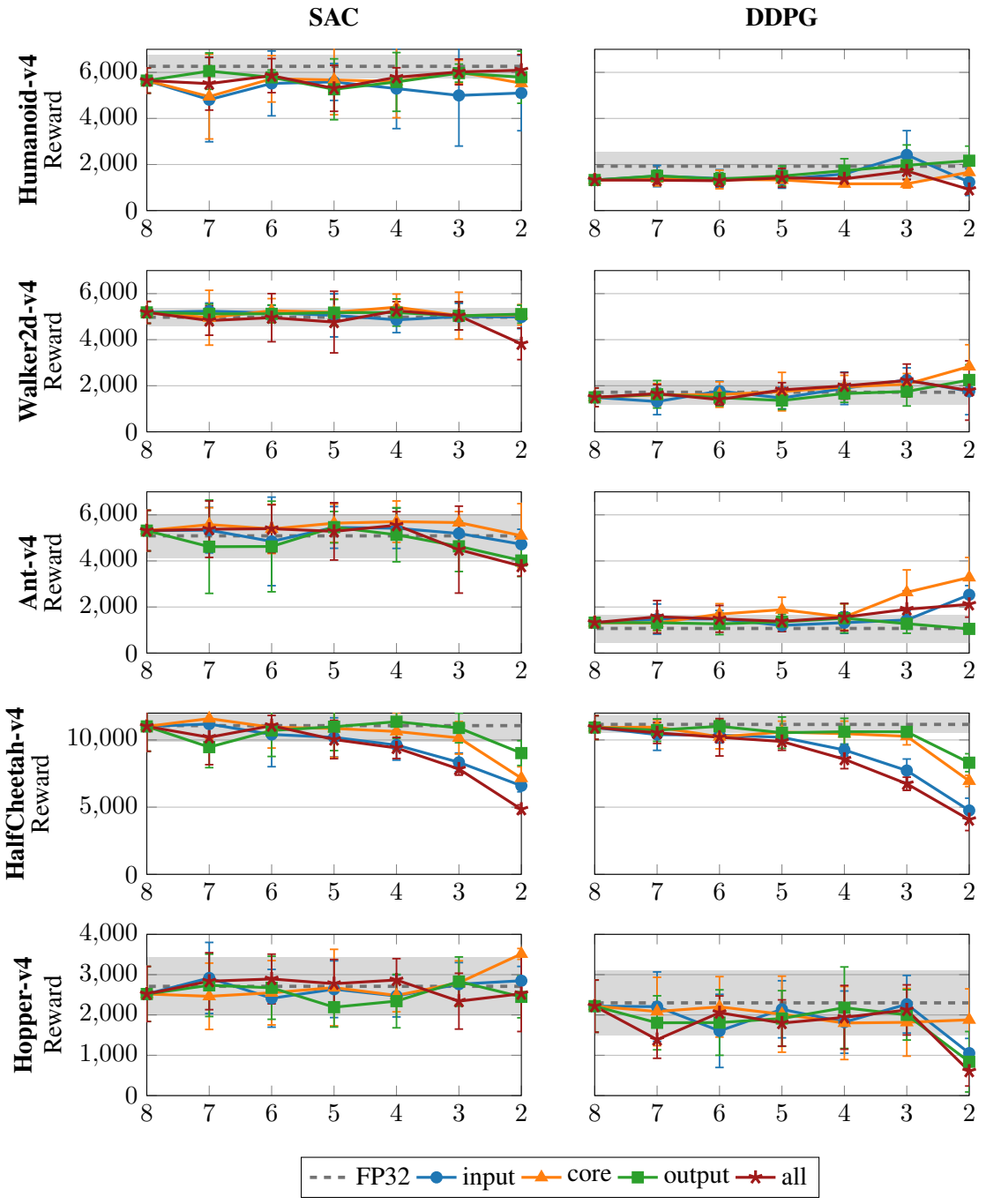

Figure 1: Reward vs. bitwidth for full-precision (FP32) baselines (shaded region indicates one standard deviation) as well as four variants of network quantization: *all*: all network operations are quantized to indicated bitwidths; *input*/*output*: the quantization of only the inputs/outputs are varied; *core*: the quantization of weights and internal activations are varied. In the latter three cases, all other components are left at 8-bit precision. We achieve FP32-parity with SAC and DDPG across most quantization scopes and environments. See main text for further discussion of the curves.

3.1. Bitwidth Sensitivity

To determine where quantization matters, we sweep bitwidth for four quantization scopes, always using QAT for all weights, activations, and biases but fixing non-swept parts to 8-bit. We investigate four scopes. *all*: all network components (input state activation, output, weights, and other activations) are quantized to varying bitwidths; *input*: only the bitwidth used to represent the input state is varied; *output*: only the final pre-hyperbolic tangent activation requantization is varied; *core*: all internal weights and activation values are quantized. Figure 1 reports returns for each scope and bitwidth; shaded bands show mean plus/minus one standard deviation across 10 models per datapoint. We consider a quantized model to *match* FP32 returns if its mean return lies within the FP32 band.

SAC For SAC, we find that our quantized models generally match the FP32 baseline down to at least 3 bits, even in the *all* setting. The only major exception is HalfCheetah, where we find that starting at 5-bits reward decreases. This is consistent with a decreasing reward in the *input* setting, indicating that input quantization alone can have a major impact on reward. However, we also find that for HalfCheetah the *core* setting for 2-bit results in a reward drop; hinting at model capacity-limited return, which is further supported by layer-width reduction results in Appendix B. For Humanoid, we observe that the variance between policies is often higher across bitwidths than the FP32 policy. However, both the *core* and *all* settings clearly match the FP32 baseline at 3-bits, with similar variances. Last, we observe that output quantization only appears to matter at the extreme of 2-bits.

DDPG With DDPG, we observe lower overall rewards, though the quantized versions generally match the FP32 baselines. HalfCheetah shows a performance drop similar to the one seen with SAC. Notably, for the Ant task, lower bit-width quantization appears to improve rewards. We hypothesize that this behavior relates to findings that RL performance does not always improve monotonically with model capacity (Bjorck et al., 2021).

For both algorithms, the *output* quantization has the least influence on performance. However, as the output quantization also has only a minor influence on the size of the synthesized hardware model, when compared to other parameters, we leave it at 8-bit for the remainder of this work.

In our environments, if FP32 parity cannot be reached in the *all* setting, the input bitwidth can generally be considered the bottleneck, as we observe the *core* and *output* setting consistently outperforming the *input* setting. We attribute this to the fact that high quantization error in the first layer prevents the network from clearly differentiating between similar sensor states, causing a loss of information important for fine control. In contrast, once the signal passes the first layer, the information is distributed across a latent space with more redundancy. This makes the deeper layers more resilient to lower bit-widths, as the network is no longer relying on the high-precision resolution of a single input feature. Similar results of the importance of the first layer have also been found, for instance, regarding sparsity in convolutional neural networks (Han et al., 2015). As SAC outperforms DDPG in our setting, we use it in the remainder of this work.

3.2. Model Selection

Guided by the sensitivity results, we select SAC configurations under the previous FP32-parity criterion (mean within the FP32 mean plus/minus one standard deviation). First, we choose the smallest core quantization precision, a single bitwidth (b_{core}) shared by hidden-layer weights and

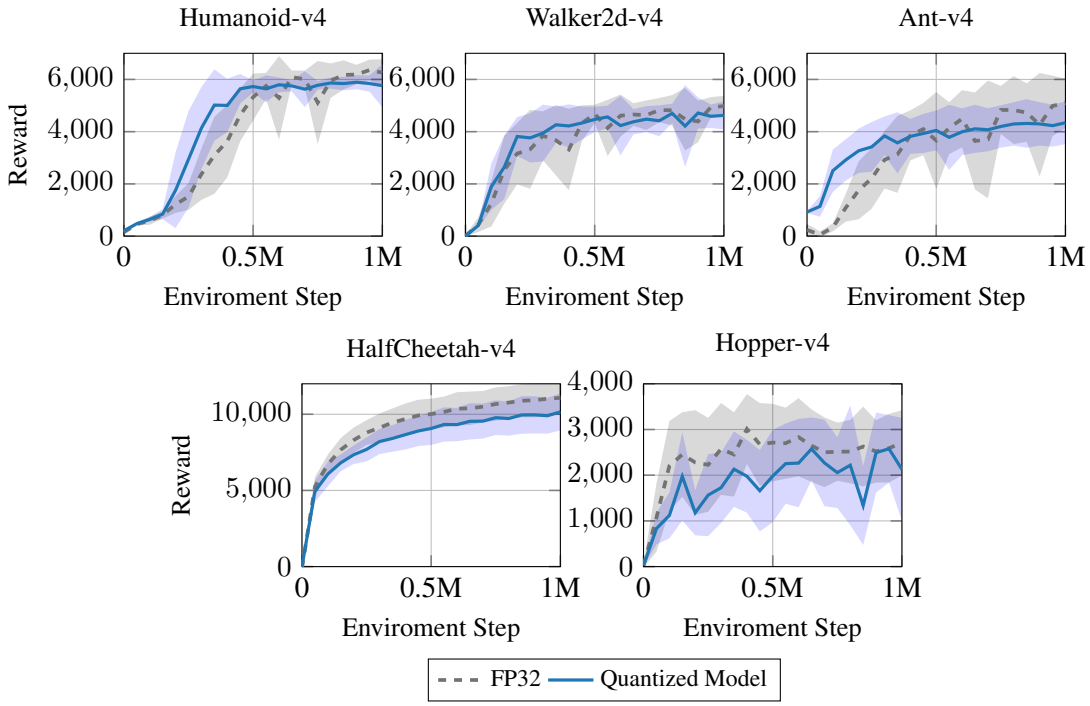

Figure 2: Evaluation reward across training time steps for our environments with SAC. Shaded bands show standard deviation over trained models. Overall, the selected quantized models show comparable convergence behavior to the floating-point baseline.

internal activations, that matches FP32 with input/output fixed at 8-bit (Figure 1). We prioritize b_{core} because it strongly affects hardware cost (e.g., requantization resources scale exponentially with activation bitwidth in FINN (Blott et al., 2018)). Next, holding that precision, we sweep the hidden layer width h over $\{256, 128, 64, 32, 16\}$ and pick the smallest that still matches FP32. Finally, with b_{core} and h fixed, we sweep input state quantization bits (b_{in}). Figures showing sweeps for model selection are available in Appendix B.

Table 1 reports the resulting models. The selected models evaluation reward over training environment steps is shown in Figure 2, with 10 rollouts performed per evaluation step, per model. With our staged selection, FP32 parity is always achieved with *3-bit cores*, often even 2 bits suffice. The tolerable hidden-width reduction and observation quantization are environment-dependent. Furthermore, the acceptable input activation bitwidth shrinks as core precision and hidden layer width is reduced: compare Figure 1 (input sweep with core fixed) to Table 1 (after fixing the minimal FP32-parity b_{core} and h), where the attainable input precision is generally lower.

Environment	h	b_{core}	b_{in}
Humanoid	16	3	4
Walker2d	128	2	3
Ant	64	2	3
HalfCheetah	256	3	8
Hopper	16	2	6

Table 1: Selected SAC policies: *hidden layer width* h; *core bitwidth* b_{core} (weights and activations); *input bitwidth* b_{in}.

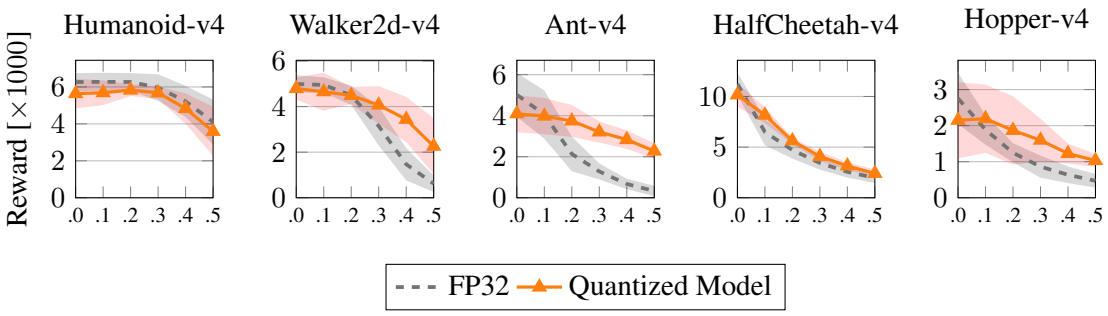

Figure 3: Robustness to observation input noise. Reward vs. noise level, σ, for floating-point and selected QAT policies on MuJoCo tasks. Shaded bands show standard deviation over trained models. The quantized, selected model performs better, or on par, with the FP32 baseline under injection.

3.3. Noise injection

We evaluate robustness to input state perturbations by adding i.i.d. Gaussian noise to the *normalized* state at inference: $\hat{s} = \mathrm{norm}(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$ with $\sigma \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. We compare the *selected* quantized model (Table 1) with the FP32 baseline. For each σ and task we average returns over 10 models (10 episodes each) and report the standard deviation of model means. Figure 3 shows return vs. σ. Across tasks, quantized policies match or exceed FP32 at higher noise levels for the quantized models. We hypothesize this to be due to training time discretization of the state space. While very similar states are often mapped to the same quantization bins, which filters out small noise, equally small noise can lead to changes in the utilized bins. To achieve high returns despite these discrete jumps for very similar states, the policy is forced to remain stable across adjacent bins. In contrast, the FP32 baseline may react to even small perturbations.

3.4. FPGA Synthesis

We synthesize policies from Table 1 for an Artix-7 XC7A15T (speed grade -1); available device resources are listed in Table 2. Note that this is a low-end FPGA, having a limited number of resources. Larger FPGAs could improve throughput and latency. For all networks, we only investigate the performance and hardware requirements for the integer portion; hence, initial floating-point quantization and final hyperbolic tangent mapping are not considered. Networks are exported with FINN and built out-of-context (OOC) as IP cores without a board wrapper. All action dimensions are padded to multiples of 32 for compatibility with FINN.

Resource	Quantity
Lookup-Tables (LUTs)	10,400
Flip-flops (FFs)	20,800
Block RAM (36 Kb)	25
Digital Signal Processing Units (DSPs)	45

Table 2: XC7A15T-FGG484-1 device resources.

Synthesis follows a throughput-driven procedure: for each policy, we sweep target throughputs in powers of 10, while FINN selects folding factors, SIMD width and the number of PEs, to approximately hit the target at a fixed 100 MHz clock. We retain the highest target whose build completes and meets timing. The resulting resources, latency and estimated power are summarized in Table 3.

8-bit models exceed XC7A15T capacity, because FINN's integer-only requantization memory cost grows exponentially with bitwidth (Blott et al., 2018). We therefore use a reference that pre-

	Environment	LUTs	FFs	BRAM	DSP	Latency	P [W]	TP	E.p.A. [J]
Selected Model	Humanoid	2.3 k	3.1 k	1.5	45	15.4 µs	0.33	6.5×10^4	5.1×10^{-6}
	Walker2D	1.9 k	1.6 k	2	4	162.2 µs	0.17	6.2×10^3	2.8×10^{-5}
	Ant	2.7 k	4.5 k	3	45	2.3 µs	0.39	4.4×10^5	8.9×10^{-7}
	HalfCheetah	4.3 k	4.6 k	15	11	243.2 µs	0.33	4.1×10^3	8.0×10^{-5}
	Hopper	2.4 k	2.0 k	0	45	0.2 µs	0.31	4.8×10^6	6.5×10^{-8}
Reference (8-4-8)	Humanoid	8.4 k	9.5 k	25	22	243.3 µs	0.58	4.1×10^3	1.4×10^{-4}
	Walker2D	5.8 k	5.7 k	25	11	243.3 µs	0.40	4.1×10^3	9.7×10^{-5}
	Ant	4.6 k	5.9 k	25	11	243.3 µs	0.37	4.1×10^3	9.0×10^{-5}
	HalfCheetah	5.3 k	5.7 k	25	11	243.3 µs	0.38	4.1×10^3	9.2×10^{-5}
	Hopper	4.4 k	5.4 k	25	11	243.3 µs	0.35	4.1×10^3	8.5×10^{-5}

Table 3: Post-synthesis resource utilization, end-to-end latency, power estimated by the Xilinx Power Estimator (P) in Watts, peak throughput (TP) in actions per second, and energy per action (E.p.A.) in Joule on a Artix-7 XC7A15T−1 at 100 MHz.

serves the original architecture (width 256) with a 4-bit *core* (I/O at 8 bit) and the prior folding selection. Except for Humanoid, these designs use similar resources as padding yields matched widths; Humanoid differs due to its larger input dimension. Latency and energy per action improve for all selected models relative to this reference except HalfCheetah, where our selection (256, 3-bit) mainly reduces BRAM usage compared to the 4-bit reference. Walker2d gains a $\sim 1.5\times$ speedup while remaining wide (128), but overall resource and power needs drop substantially. Ant and Humanoid exhibit order-of-magnitude latency gains ($10\times$–$100\times$), and Hopper reaches a 21-cycle latency, corresponding to a $\sim 1000\times$ improvement. Note that a 4-bit reference is already a strong baseline; prior work (Krishnan et al., 2022) reported collapse on Walker2d at 4-bit quantization.

4. Related Work

Quantized training for RL. The most closely related works are Krishnan et al. (2022); Lu et al. (2024) and Ivanov et al. (2025), who also study QAT for MuJoCo tasks. Of these, only Krishnan et al. (2022) investigates QAT for low bitwidths in MuJoCo, as we do here. Specifically, they study both post-training quantization (PTQ) in a distributed actor-learner setup, and QAT on Atari and some MuJoCo tasks for different RL methods, including DDPG but not SAC. Unfortunately, their public implementation does not cover the low-bitwidth experiments, and key QAT details are missing from the manuscript. Known setting differences include a much larger budget of environment steps (10M steps), where QAT is enabled only after 5M steps, whereas we train for 1M steps with QAT from scratch. Furthermore, the environment versions differ. Therefore, only a qualitative comparison with their results makes sense: Krishnan et al. (2022) report QAT experiments for HalfCheetah and Walker2d with DDPG. For both tasks, they report substantial return degradation starting at 4 to 5 bits. In our work, the *core* setting for these tasks shows reward degradation only at 2 bits, or none, respectively (Figure 1, DDPG). Note that the baseline reward they obtain with Walker2d is comparable to ours, whereas our baseline obtains much higher returns for HalfCheetah.

Lu et al. (2024) find that 8-bit QAT under their protocol performs worse than FP32 networks on MuJoCo tasks. Ivanov et al. (2025) pretrain in full precision and then fine-tune with QAT for 8-bit *weights* while keeping *activations* in FP32, thereby avoiding activation quantization error but

also limiting end-to-end efficiency gains. Chandrinos et al. (2024) study both PTQ and QAT in a multi-agent environment with discrete actions. They find that QAT substantially outperforms PTQ.

Gil et al. (2021) study PTQ for continuous control by quantizing and pruning pretrained policies. They find that 4-bit weights largely preserve return on a real inverted pendulum setup, whereas 3-bit and below resulted in a substantial loss of reward.

Pruning for RL. Pruning—removing parameters or connections—can reduce multiply-accumulate (MAC) operations and memory traffic at inference. In reinforcement learning, very high levels of unstructured sparsity (often $> 95\%$) have been reported without return loss across diverse settings (Ivanov et al., 2025; Tan et al., 2023; Arnob et al., 2024; Graesser et al., 2022), and scaling studies suggest that larger, and pruned, policies can even improve reward scaling (Ma et al., 2025). Pruning has also been studied in combination with QAT (Lu et al., 2024; Ivanov et al., 2025). However, unstructured sparsity is generally difficult to exploit in hardware, and we therefore do not consider it in our work. Nevertheless, these works show that standard RL networks are typically severely overparameterized, inspiring our architecture search.

Related Directions. At the quantization extreme, work on binarized policies targets discrete action spaces (Kadokawa et al., 2021; Lazarus and Kochenderfer, 2022; Valencia et al., 2019), with some studies binarizing only parts of the network (Chevtchenko and Ludermir, 2021). Additionally, several works discretize only parts of the RL pipeline for practical reasons: discretizing the continuous action space to change the algorithmic regime (Dadashi et al., 2022), or quantizing inputs to reduce replay-buffer memory with negligible return loss (Grossman and Plancher, 2023). Last, performing training in half-precision has been explored (Björck et al., 2021). These target different bottlenecks than end-to-end integer inference.

5. Discussion & Future Work

In this work, we show how to use quantization-aware training to obtain deployment-ready, integer-only policy networks that match the returns of full-precision networks on MuJoCo tasks. A key finding of our work is that network weights and internal activations can be quantized quite aggressively (a bitwidth of 2 or 3 suffices), allowing an implementation on low-resource FPGA hardware with sub-millisecond inference latency and consuming microjoules per action inference. Furthermore, *input* quantization should be treated separately, because it can have a major influence on the policy quality. We also find that quantized models are often more robust under input noise, presumably because the noisier gradient updates during training act as implicit regularization. We therefore hypothesize that QAT-trained policies may be more robust in the real world under sensor noise.

Despite the promising results, a number of limitations and open questions remain. First, our study reflects the situation of training and deployment in simulation only. Clearly, for real-world systems, additional complications can emerge. For example, aspects such as safety and security need to be taken into account besides the expected reward that we use in our analysis to judge policy quality. Also, real-world noise can have different and more challenging characteristics than the analytically injected noise in our experiments.

For future work, it would be interesting to move beyond classical MuJoCo tasks and investigate additional scenarios, such as vision-based policies for mobile robots or sparse rewards. An additional direction is evaluation on different hardware targets (other FPGAs and non-FPGA platforms) and on physical systems to capture sensor/actuator overheads and measured power.

Acknowledgments

This work was partially supported by the Austrian Science Fund (FWF) [10.55776/COE12]. This work was supported in part by the ERC-2020-AdG 101020093 (VAMOS). The research was supported by the Scientific Service Units (SSU) of ISTA through resources provided by Scientific Computing (SciComp).

References

Samin Yeasar Arnob, Riyasat Ohib, Sergey Plis, Amy Zhang, Alessandro Sordoni, and Doina Precup. Efficient reinforcement learning by discovering neural pathways. *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.342*, 2013.

Johan Björck, Xiangyu Chen, Christopher De Sa, Carla P Gomes, and Kilian Weinberger. Low-precision reinforcement learning: Running soft actor-critic in half precision. In *International Conference on Machine Learning (ICML)*, 2021.

Nils Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning with spectral normalization. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2018.

Nikolaos Chandrinos, Michalis Amasialidis, Manos Kirtas, Konstantinos Tsampazis, Nikolaos Passalis, and Anastasios Tefas. Quantization-aware training for multi-agent reinforcement learning. In *European Signal Processing Conference (EUSIPCO)*, 2024.

Sérgio F Chevtchenko and Teresa B Ludermir. Combining STDP and binary networks for reinforcement learning from images and sparse rewards. *Neural Networks*, 2021.

Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous control with action quantization from demonstrations. In *International Conference on Machine Learning (ICML)*, 2022.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 2022.

Rika Sugimoto Dimitrova, Mathias Gehrig, Dario Brescianini, and Davide Scaramuzza. Towards low-latency high-bandwidth control of quadrotors using event cameras. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

Giuseppe Franco, Alessandro Pappalardo, and Nicholas J Fraser. Xilinx/brevitas, 2025. URL https://doi.org/10.5281/zenodo.3333552.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision (LPCV)*. 2022.

Yoonhee Gil, Jong-Hyeok Park, Jongchan Baek, and Soohee Han. Quantization-aware pruning criterion for industrial applications. *IEEE Transactions on Industrial Electronics*, 2021.

Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2022.

Lev Grossman and Brian Plancher. Just round: Quantized observation spaces enable memory efficient learning of dynamic locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research (JMLR)*, 2022.

Dmitry A Ivanov, Denis A Larionov, Oleg V Maslennikov, and Vladimir V Voevodin. Neural network compression for reinforcement learning tasks. *Scientific Reports*, 2025.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Yuki Kadokawa, Yoshihisa Tsurumine, and Takamitsu Matsubara. Binarized p-network: Deep reinforcement learning of robot control from raw images on FPGA. *IEEE Robotics and Automation Letters*, 2021.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning (CORL)*, 2018.

Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 2015.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 2023.

Srivatsan Krishnan, Maximilian Lam, Sharad Chitlangia, Zishen Wan, Gabriel Barth-Maron, Aleksandra Faust, and Vijay Janapa Reddi. QuaRL: Quantization for fast and environmentally sustainable reinforcement learning. *Transactions on Machine Learning Research (TMLR)*, 2022.

Christopher Lazarus and Mykel J. Kochenderfer. Deep binary reinforcement learning for scalable verification. *arXiv preprint arXiv:2203.05704*, 2022.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Heng Lu, Mehdi Alemi, and Reza Rawassizadeh. The impact of quantization and pruning on deep reinforcement learning models. *arXiv preprint arXiv:2407.04803*, 2024.

Guozheng Ma, Lu Li, Zilin Wang, Li Shen, Pierre-Luc Bacon, and Dacheng Tao. Network sparsity unlocks the scaling potential of deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2025.

NVIDIA. NVIDIA H100 Data Sheet, 2025. URL https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306. Accessed: 2025-11-06.

Daniele Palossi, Nicky Zimmerman, Alessio Burrello, Francesco Conti, Hanna Müller, Luca Maria Gambardella, Luca Benini, Alessandro Giusti, and Jérôme Guzzi. Fully onboard AI-powered human-drone pose estimation on ultralow-power autonomous flying nano-UAVs. *IEEE Internet of Things Journal*, 2021.

Gulnaz Rati, Rafael Mendes, and Aisha Noor. Adaptive dataflow and precision optimization for deep learning on configurable hardware architectures. 2025.

Erik Schuitema, Lucian Buşoniu, Robert Babuška, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics Science and Systems (RSS)*, 2018.

Yiqin Tan, Pihe Hu, Ling Pan, Jiatai Huang, and Longbo Huang. RLx2: Training a sparse deep reinforcement learning model from scratch. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

Raul Valencia, Chiu-Wing Sham, and Oliver Sinnen. Using neuroevolved binary neural networks to solve reinforcement learning environments. In *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2019.

Zishen Wan, Bo Yu, Thomas Yuang Li, Jie Tang, Yuhao Zhu, Yu Wang, Arijit Raychowdhury, and Shaoshan Liu. A survey of FPGA-based robotic computing. *IEEE Circuits and Systems Magazine*, 2021.

Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and FrankaPy. *arXiv preprint arXiv:2011.02398*, 2020.

Appendix A. Hyperparameters

Default models use the ReLU activation function and a hidden layer width of 256 neurons.

Table 4: SAC hyperparameters used in our experiments.

Hyperparameter	**Value**	**Notes**
Total timesteps	1,000,000	Environment steps during training.
Replay buffer size	1×10^6	Transitions stored.
Discount γ	0.99	Future reward discount.
Target smoothing τ	0.005	Soft target update rate.
Batch size	256	Minibatch size for updates.
Learning starts	5×10^3	Steps collected before learning.
Policy LR	3×10^{-4}	Adam LR for policy.
Q-network LR	1×10^{-3}	Adam LR for critics.
Policy update frequency	2	Update policy every 2 critic steps.
Target network frequency	1	Target update every critic step.
Entropy	autotune	Adapted online.

Table 5: DDPG hyperparameters used in our experiments.

Hyperparameter	**Value**	**Notes**
Total timesteps	1,000,000	Environment steps during training.
Learning rate	3×10^{-4}	Optimizer step size (actor & critic).
Replay buffer size	1×10^6	Transitions stored.
Discount γ	0.99	Future reward discount.
Target smoothing τ	0.005	Smoothing for target nets.
Batch size	256	Minibatch size for critic/actor updates.
Exploration noise (std)	0.1	Gaussian action noise during data collection.
Learning starts	2.5×10^4	Steps collected before learning.
Policy update frequency	2	Update policy every 2 critic steps.

Appendix B. Model Selection

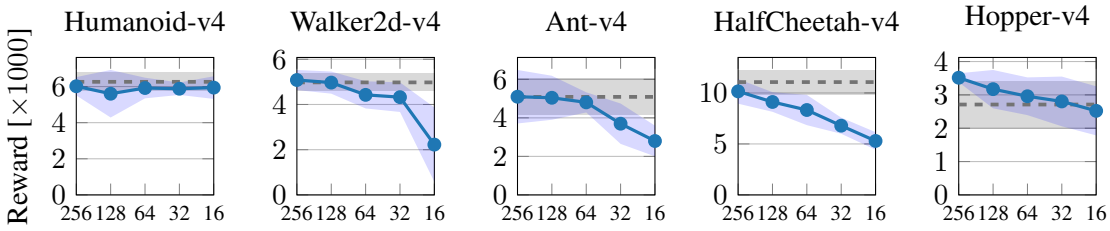

Figure 4: Return vs. hidden width for SAC under the minimal FP32-matching **core** precision (2-bit except 3-bit for HalfCheetah/Humanoid). FP32 mean and its one-standard deviation band shown for reference.

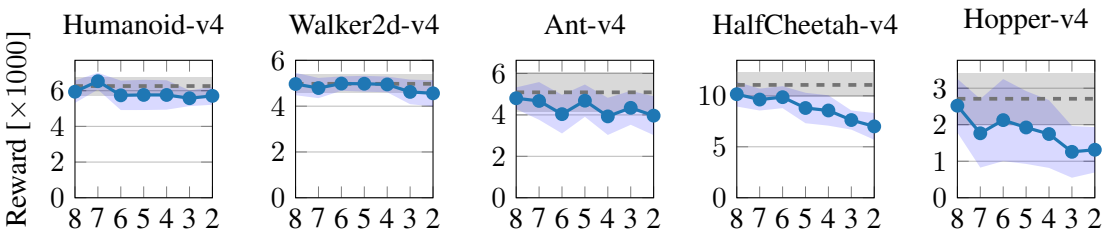

Figure 5: Return vs. input quantization for SAC under the configuration from Table 1, except input bits, which is swept here. FP32 mean and its one-standard deviation band shown for reference.

Appendix C. Input Normalization

SAC For SAC, we find that running input normalization improves returns for the floating-point baseline (except HalfCheetah, but results are close with standard deviations overlapping); see Table 6. Due to this, and SAC's better overall returns, when compared to DDPG, we generally apply normalization across this work. We hypothesize that input normalization improves returns, specifically in the quantization context, because it improves optimization of the quantization scale parameter in the first layer, which we show to have major impact on returns in Section 3.

Environment	No Input Normalization	Input Normalization
Ant	$5.0\,\mathrm{k} \pm 1.7\,\mathrm{k}$	$5.1\,\mathrm{k} \pm 0.9\,\mathrm{k}$
Hopper	$2.4\,\mathrm{k} \pm 1.0\,\mathrm{k}$	$2.7\,\mathrm{k} \pm 0.7\,\mathrm{k}$
HalfCheetah	$11.6\,\mathrm{k} \pm 1.0\,\mathrm{k}$	$11.1\,\mathrm{k} \pm 1.2\,\mathrm{k}$
Walker2d	$4.3\,\mathrm{k} \pm 0.9\,\mathrm{k}$	$5.0\,\mathrm{k} \pm 0.4\,\mathrm{k}$
Humanoid	$4.7\,\mathrm{k} \pm 1.4\,\mathrm{k}$	$6.3\,\mathrm{k} \pm 0.5\,\mathrm{k}$

Table 6: Floating point baseline, with and without input normalization. Mean and standard deviation over 10 trained models and 1000 rollouts per model. The floating point baseline with input normalization performs on par or better than without normalization.

DDPG Figure 6 shows a floating point baseline and various quantization scopes trained with DDPG, as described in Section 3. The results for the left-hand side plots were obtained *without* input normalization (as implemented in CleanRL), whereas we use running, per-dimension input normalization for the right-hand side plots. While the floating-point baseline with running mean performs worse than its non-normalized counterpart, we observe the opposite behaviour for the quantized networks. Importantly, the quantized networks *with* input normalization consistently perform on par with the floating-point networks *without* input normalization (which are the stronger baselines).

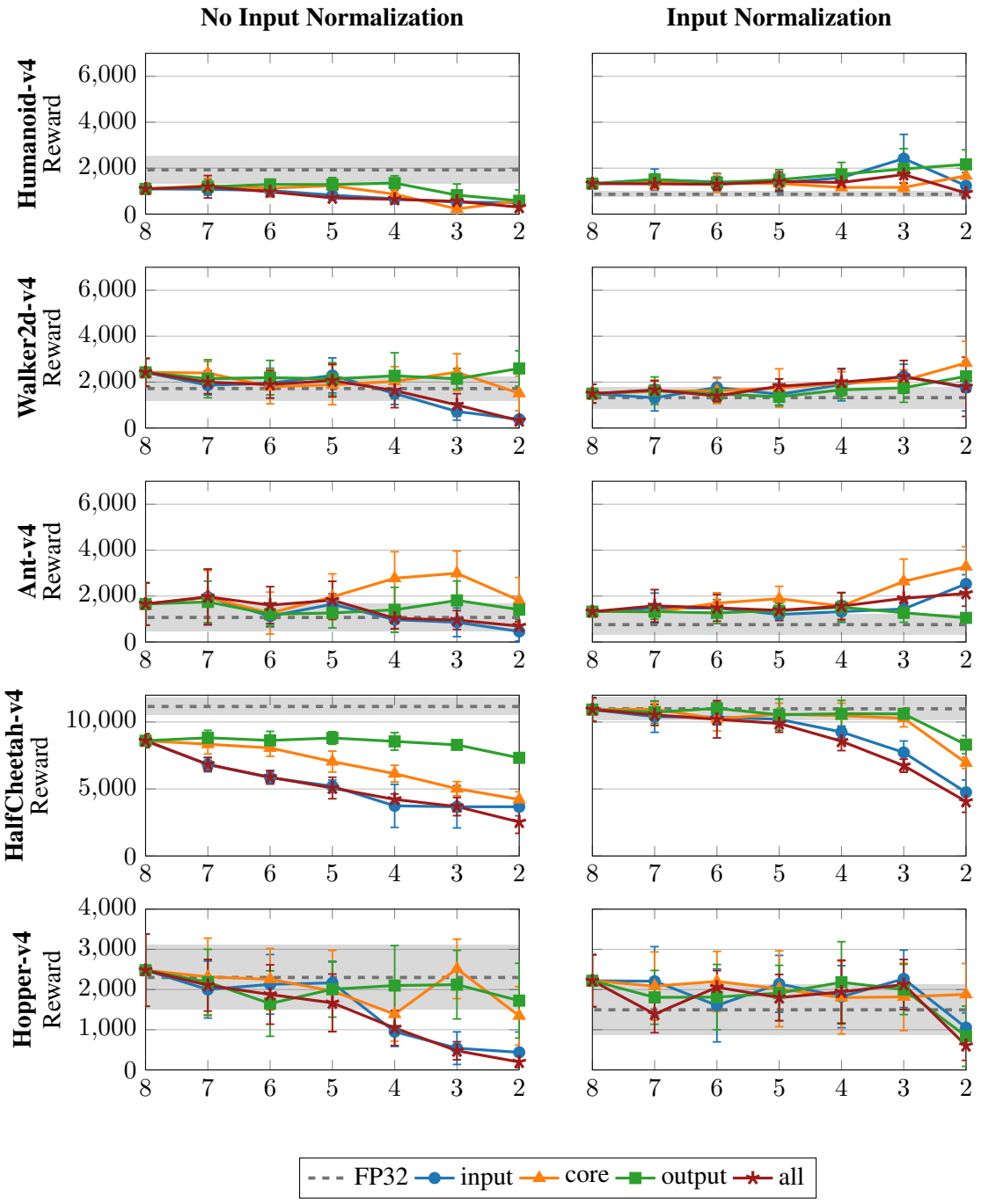

Figure 6: DDPG; Reward vs. bitwidth for full-precision (FP32) baselines (shaded region indicates one standard deviation) as well as four variants of network quantization: *all*: all network operations are quantized to indicated bitwidths; *input*/*output*: the quantization of only the inputs/outputs are varied; *core*: the quantization of weights and internal activations are varied. In the latter three cases, all other components are left at 8-bit precision. We achieve FP32-parity on most task bitwidth combinations with DDPG when we utilize input normalization.