# Experiences with VITIS AI for Deep Reinforcement Learning

Nabayan Chaudhury
Department of CS
Virginia Tech
Blacksburg, VA, USA
nabayanc@vt.edu

Atharva Gondhalekar
Department of ECE
Virginia Tech
Blacksburg, VA, USA
atharva1@vt.edu

Wu-chun Feng
Department of CS and ECE
Virginia Tech
Blacksburg, VA, USA
wfeng@vt.edu

***Abstract*—Deep reinforcement learning has found use cases in many applications, such as natural language processing, self-driving cars, and spacecraft control applications. Many use cases of deep reinforcement learning methodologies impose additional restrictions on the underlying hardware that runs policy evaluation algorithms. For example, in many mission-critical systems, the deployment of reinforcement learning often requires inference with low latency and high accuracy. Recent studies in machine learning have explored the impact of pruning the trained network and shown that pruning the network can improve the accuracy and performance of the inference. Inspired by the recent studies on the effects of pruning trained networks, this work explores the efficacy of the network pruning technique via the AMD Vitis AI toolchain for reinforcement learning models.**

**In particular, we evaluate the soft actor-critic (SAC) model that is trained to solve the MuJoCo humanoid environment, where the objective of the humanoid agent is to learn a policy that allows it to stay in motion for as long as possible without falling over. During the training phase, we prune the model using the weight sparsity pruner from the Vitis AI optimizer at different timesteps. Our experiments show that pruning leads to an improvement in the reinforcement learning policy evaluation, where the trained agent can remain mobile in the environment and accumulate higher rewards compared to a trained agent without pruning. Specifically, we observe that pruning the network during training can deliver up to 23% higher reward and 20% better mean episode length compared to a network without any pruning.**

***Index Terms*—reinforcement learning, humanoid, MuJoCo, network pruning, FPGA, GPU, Vitis AI**

## I. Introduction

In recent years, deep reinforcement learning (DRL) has found use cases in applications such as self-driving cars [1], natural language processing [2], and mission-critical tasks such as landing a spacecraft on celestial bodies [3]. Low-latency decision-making while maintaining the quality of solutions in resource and power-constrained environments is critical to the success of DRL algorithms in many applications, making fast and accurate DRL policy evaluation desirable.

Recent studies in the area of machine learning (ML) have explored the impact of pruning the trained network on the accuracy and performance of inference [4]–[6]. The process of pruning neural networks typically involves removing the connections between network layers, based on a pruning policy. Pruning has been shown to reduce the amount of computation necessary to generate the output of the inference, and in many cases, maintain the accuracy of the inference [4], [5]. Figure 1 shows an example of pruning a neural network using Vitis AI [7], a software stack developed by AMD for neural network inference on field-programmable gate arrays (FPGAs). In this work, we evaluate the efficacy of a weight sparsity pruner from the Vitis AI toolchain for the DRL model. We evaluate the soft actor-critic (SAC) model that is trained to solve the MuJoCo humanoid environment [8], where the humanoid agent learns a policy with the objective of remaining in a healthy state (i.e., remaining in motion) for as long as possible without falling over. We analyze the impact of pruning the network at various stages of the training phase and measure the performance of pruned networks using quantitative measures such as mean reward and mean episode length. We evaluate the performance impact of model pruning on an Nvidia RTX 3090 GPU, providing us insight into model pruning for DRL networks.

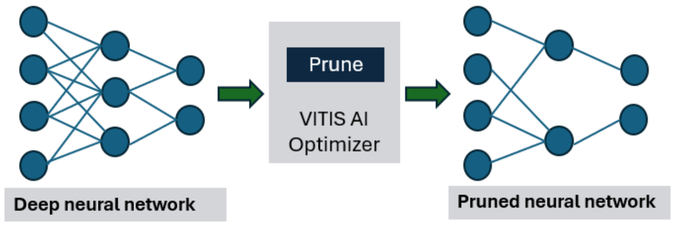


Fig. 1: Pruning a deep neural network using Vitis AI

In all, we make the following contributions in this paper.

- Application of Vitis AI to prune DRL networks for higher reward and better mean episode length.
- Rigorous evaluation of the efficacy of pruning the trained model at various stages during the training phase.
- Up to 23% higher reward and 20% higher (i.e., better) mean episode length compared to the neural network without any pruning.

The rest of the paper is organized as follows: §II outlines related work on DRL, neural network pruning, and Vitis AI. §III describes the off-policy DRL along with simulated physics

The work detailed herein has been supported in part by NSF I/UCRC CNS-1822080 via the NSF Center for Space, High-performance, and Resilient Computing (SHREC).

environments and the scope of Vitis AI. §IV articulates the MuJoCo humanoid environment. §V presents our evaluation on the the impact of pruning during the training and policy evaluation phases. Finally, §VI describes future directions for this work while §VII provides a conclusion to this work.

## II. Related Work

We present related work in three parts: (1) reinforcement learning methods for the MuJoCo humanoid environment, (2) prior work on exploring the effects of pruning in machine learning, and (3) existing studies that make use of Vitis AI.

### A. Reinforcement Learning (RL) in MuJoCo Environment

Multi-joint dynamics with contact, or MuJoCo for short, is a general-purpose physics environment developed by Google-Deepmind [9]. This work focuses on the humanoid environment within MuJoCo, where the objective of the humanoid agent is to remain in a healthy state characterized by a continuous state of motion without falling over. The MuJoCo environment has been extensively used in machine-learning (ML) research. Wen et al. [10] use the MuJoCo environment to evaluate their multi-agent transformer network that casts multi-agent reinforcement learning (RL) into a sequence modeling problem. Shao et al. [11] present design-space exploration techniques for hardware acceleration of RL policy training and evaluate their hardware accelerator using MuJoCo environments. Liang et al. [12] present GPU-accelerated RL simulations using the MuJoCo environment.

### B. Pruning in Machine Learning

Hoefler et al. [4] survey many approaches to prune neural networks and explore multiple training strategies to achieve model sparsity. Chaturvedi et al. [5] explore the effects of introducing sparsity in a densenet and deconvolution network (DDNet). Obando-Ceron et al. [6] demonstrate that by removing network parameters during reinforcement learning (RL) policy training, it is possible to perform policy evaluations with better accuracy than evaluations with dense network counterparts. Inspired by the aforementioned studies, this work evaluates the impact of pruning DRL networks using Vitis AI.

### C. Vitis AI

Vitis AI is a software stack developed by AMD for accelerating artificial intelligence inference on AMD FPGAs [7]. Ushiroyama et al. [13] use Vitis AI to implement convolutional neural network on FPGAs. Cabrera et al. [14] use Vitis AI for performing errant beam detection on AMD FPGAs.

Our work differs from these prior studies in the following ways. First, while Vitis AI is primarily focused on optimizing the trained networks for FPGAs, we show and evaluate our pipeline that integrates Vitis AI components with a GPU implementation. Second, while the effects of pruning, the use of Vitis AI, and DRL in the MuJoCo environment are individually well-explored subjects, we evaluate the multiplicative impact of pruning DRL networks for MuJoCo environments using Vitis AI.

## III. Background

### A. Off-Policy Reinforcement Learning and Soft Actor-Critic

Reinforcement learning (RL) is a branch of machine learning (ML), where an agent learns to make optimal decisions by interacting with an environment and with the goal of maximizing a cumulative *reward* [15]. Deep reinforcement learning (DRL) combines RL with deep learning (DL), allowing deep neural networks to learn the policy from instances of the environment without constructing a complete state space of the environment. The general training mechanism in RL involves the agent making decisions in the environment, receiving rewards, and updating its policy to maximize cumulative rewards over time. Formally, the reinforcement learning (RL) problem can be defined as a policy search in a Markov decision process (MDP) defined by a tuple $(S, A, P, R, \gamma)$, where $S$ is the set of states, $A$ is the set of actions, $P$ represents the state transition probabilities, $R$ is the reward function, and $\gamma$ is the discount factor. The objective of the agent is to learn a policy $\pi(a \mid s)$ that maximizes the expected cumulative reward, defined as the return $G_t$:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

where $R_{t+k+1}$ is the reward received $k+1$ steps after time $t$, and $\gamma \in [0, 1)$ is the discount factor that determines the importance of future rewards. The goal is to find the optimal policy $\pi^*$ that maximizes the expected return from any initial state $s_0$:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[G_t \mid \pi] \tag{2}$$

State- and action-value functions estimate the expected return, and these three sets of equations govern the learning objective. The state-value function $V_\pi(s)$ and the action-value function $Q_\pi(s, a)$ are defined as:

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \tag{3}$$

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{4}$$

As the agent spends time performing actions in the environment, it updates its policy and value estimates iteratively as a coupled optimization objective.

RL algorithms can be designed in several ways. *Online reinforcement learning* involves continuous interaction of the agent with the environment, which allows the agent to learn and update its policy in real time. This enables the agent to adapt quickly to environment changes. Examples include Q-learning (DQN) [16] and actor-critic algorithms [17], [18]. *Offline reinforcement learning*, in contrast, utilizes pre-collected data to train the agent, where the agent has no further interactions with the environment. Offline algorithms, naturally can run into suboptimal policy performance when the agent encounters novel states [19]. Notable offline algorithms include conservative Q-learning (CQL) [20] and batch-constrained deep Q-learning (BCQN) [21].

In addition, algorithms can be either *on-policy* or *off-policy*. On-policy algorithms, like proximal policy optimization (PPO) [22] update the policy based on actions taken by the current policy. These methods require consistent interaction with the environment to get accurate policy updates, but they have a tendency to explore the environment less as they rely on the current policy's knowledge of the environment. In contrast, off-policy algorithms learn from actions outside the current policy, encouraging exploration and improving sample efficiency as the agent can learn from a diverse set of actions [18], [23], [24].

Our choice of algorithm is motivated by our interest in real-world, physics-based scenarios with large action spaces and complex environments, for which online, off-policy, and model-free algorithms are particularly advantageous. This led us to select soft actor critic (SAC) [18] for our experiments. SAC maximizes a tradeoff between the expected reward and the entropy of the policy, which promotes exploration and prevents premature convergence to suboptimal policies.

## B. Simulated Physics Environments

Once the RL algorithms have been trained, it is essential to have a method for benchmarking them in a controlled, reproducible setting without interacting with the real world. Such environments are generally dynamical systems with complex control laws, like *rigid body* dynamical systems (objects that do not deform and interact through collisions and forces) and *articulated body* dynamical systems (interconnected rigid bodies that exhibit human or animal motion). These systems have their own *action space* (which is a superset of the policy action space $A$) and *observation space* (which is a superset of the policy state space $S$), where:

- Action space of the system ($A_S$): set of all possible actions an agent can take in the environment.
- State space of the system ($S_S$): set of all possible configurations of the environment.

*Continuous* action spaces consist of a range of values in a given interval as control inputs and closely emulate real-world, physics-based application scenarios. For example, the $\underline{\text{Mu}}$lti-$\overline{\text{jo}}$int dynamics with $\underline{\text{co}}$ntact (MuJoCo) [25] engine provides several such articulated rigid body continuous control environments that are of interest to this study. *Discrete* action spaces consist of a finite set of actions, where each action is represented as a single binary choice, making the space lower-dimensional. This simplifies the decision-making process but may not capture the complexity of actions required in a more sophisticated environment [26].

## C. Vitis AI

Figure 2 describes Vitis AI, a comprehensive development stack developed by AMD for accelerating artificial intelligence (AI) inference on AMD hardware, including FPGAs (field-programmable gate arrays) and SoCs (system on a chip) [7]. It is designed with efficiency and acceleration in mind, allowing users to deploy accelerated machine learning models on AMD FPGAs. It provides an end-to-end solution — from model optimization to deployment — where users can get high performance, low latency, and efficient resource utilization. Figure 2 shows the following key components of Vitis AI (or VAI for short):

- *VAI Model Zoo*: A set of pre-trained and optimized models that can be easily deployed on AMD FPGA devices.
- *VAI Optimizer*: A tool capable of performing pruning on AI models, reducing model size, and improving inference speed without affecting accuracy. This paper focuses on this particular tool.
- *VAI Quantizer*: A tool that quantizes AI models by reducing precision from `fp32` (single-precision floating point) to lower precision formats like `int8`, accelerating computation and reducing memory usage.
- *VAI Compiler*: A tool that compiles the reduced model into a deployable model supported by AMD FPGA hardware, e.g., AMD Deep-Learning Processing Unit (DPU) [7].
- *VAI Profiler*: An application-level tool that allows for thorough profiling of deployable models for inference, giving insights into further optimizations for maximum performance.
- *VAI Library*: A library of APIs for easy utilization of the VAI software stack. It supports models in the Model Zoo as well as custom models.
- *VAI Runtime*: A runtime environment that executes compiled models on AMD FPGA hardware and capable of efficient resource allocation and task scheduling.
- *Deep-Learning Processing Units (DPUs)*: General-purpose AI inference accelerators that target convolution neural networks (CNNs), allowing for simultaneous deployment and inference.

Vitis AI provides a comprehensive software platform for deploying AI models for inference and supports trained models in both PyTorch and TensorFlow. The deployment pipeline typically follows the optimization step, where the optimizer prunes the network and the quantizer reduces precision, increasing memory usage and inference speed. Then, the *Model Inspector* and *Compiler* allow users to manually modify details of the model to ensure successful deployment on the DPUs, following which the runtime system allows for efficient inference. In short, Vitis AI provides the following benefits: (1) improved model inference while adhering to hardware restrictions, (2) scalability, as it supports a range of hardware from low-power devices to datacenter accelerators, ajd (3) compatibility with different deep-learning frameworks and model architectures.

To date, Vitis AI has been used to implement convolutional neural networks [29], accelerate existing object detection algorithms [30], and perform embedded object detection with custom model architectures [31]. In all cases, the reported results highlight a significant decrease in power consumption and increase in throughput. Additionally, model quantization has been shown to increase robustness against adversarial ex-

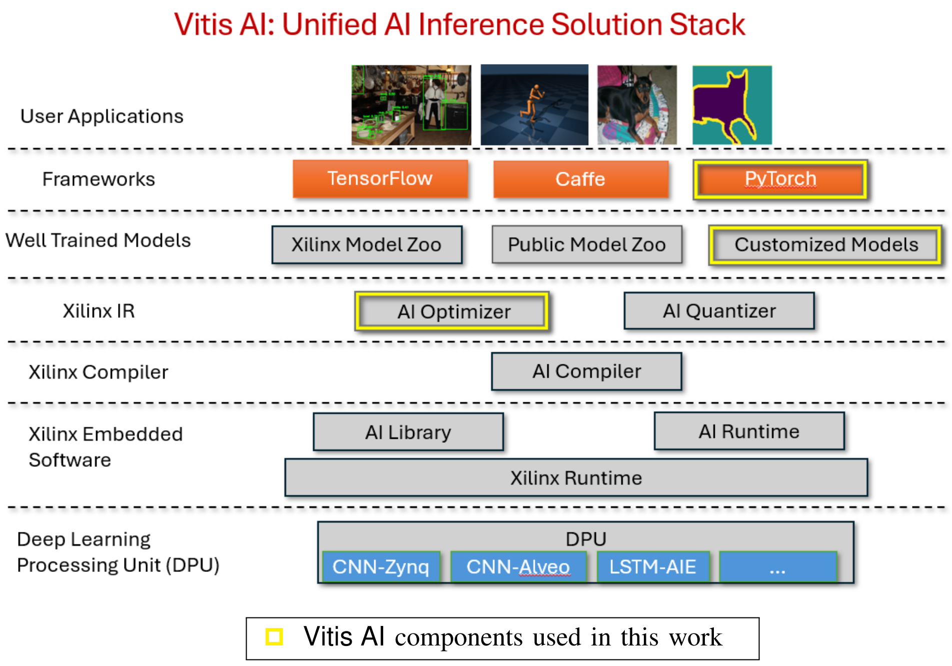


Fig. 2: Vitis AI: Unified AI inference solution stack [7], [27], [28]

amples [32], and the Vitis AI development stack has been used in real-world errant beam detection, showing fast performance and high accuracy [33].

## IV. Case Study: MuJoCo Humanoid

In this section, we use the "multi-joint dynamics with contact" environment (i.e., MuJoCo) as our case study to illustrate the efficacy of Vitis AI.

### A. Environment

The MuJoCo humanoid environment simulates a 3D bipedal humanoid robot designed to mimic human locomotion [34]. Figure 3 shows the humanoid agent in different states that it can end up in through an episode of training [8]. The robot consists of a torso with a pair of legs and arms. Each leg has three segments (thigh, shin and foot), and each arm has two segments (upper and lower arm).

The primary goal in this environment is to control the humanoid to walk or run as far as possible without falling. The following summarizes the action and state spaces:

- **Action Space**:
  - *Type*: Continuous
  - *Dimension (Degrees of freedom)*: 17
  - *Range*: Box(-0.4, 0.4, (17,), float32)
  - *Description*: The action space is defined by the torques acting at each of the humanoid's joints
- **Observation Space**:
  - *Type*: Continuous
  - *Dimension*: 376
  - *Range*: Box(-inf, inf, (376,), float64)
  - *Description*: The observation space consists of the position and velocity values of the various joints and body parts. These in turn define the state the humanoid is in at any given time.

The reward structure in the humanoid encourages stable and efficient locomotion. It is divided into a *healthy reward* that is a fixed reward for every timestep that the humanoid remains standing, a *forward reward* that is calculated based on the forward displacement of the humanoid's center of mass and encourages forward movement, a *control cost* penalty for using excessive control forces, and a *contact cost* penalty for high external forces. This naturally translates to locomotion being human-like and not forced by excessive inputs. Each episode terminates when either the humanoid reaches 1,000 timesteps, or the humanoid becomes *unhealthy*, signified by the z-coordinates of the torso falling outside a healthy range. Because the goal of the agent is to keep walking as long as possible, performance evaluation can be done by measuring the *average episode lengths* and *average cumulative rewards*. If the episodes run for longer timesteps, the cumulative reward increases and the agent is capable of walking further.

### B. Algorithm

For the purpose of our experiments, we use the Stable-Baselines 3 (SB3) library and its implementation of the Soft Actor-Critic Algorithm (SAC) [18], [35]. SB3 provides a comprehensive set of algorithm implementations for fast development and deployment, while supporting custom policy networks and in-house benchmarking and evaluation. We choose SAC because of its model-free, off-policy, and online nature, making it ideal for solving large, physics-based environments

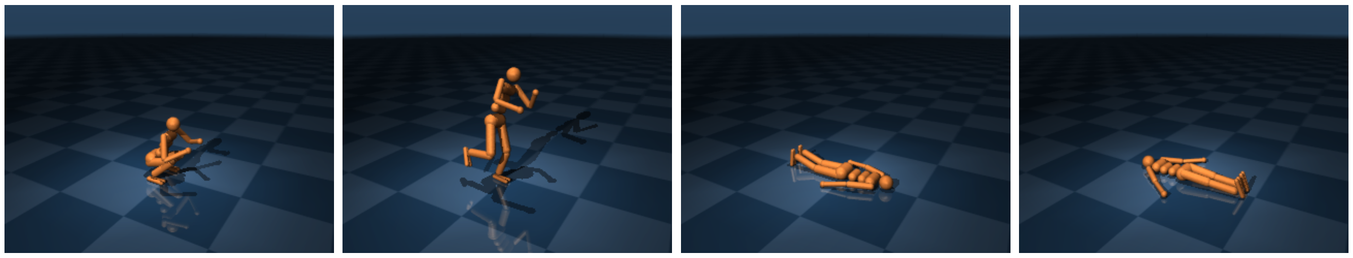

(a) Squatting (b) Running (c) Prone (d) Supine

Fig. 3: MuJoCo [8], [9] humanoid exhibiting different poses

that require exploration to reach an optimal policy. We train the agent using SAC for a total of 100,000 timesteps in each experiment, pruning the model after we reach 10%, 20%, 50%, 75% and 100% of training timesteps. Pruning is done using the Sparse Pruner from the Vitis AI Optimizer, setting the weight sparsity to 0.5. The complete details of the algorithm can be found in Algorithm 1.

**Algorithm 1:** Training and pruning SAC model on MuJoCo humanoid with Vitis AI

```
Input  : Total timesteps TOTAL_TIMESTEPS,
         Pruning timestep PRUNING_TIMESTEP,
         Number of evaluation episodes
         NUM_EVAL_EPISODES
Output: Trained and pruned SAC model, Evaluation
         results
Data: Training and evaluation environments, SAC
      model, Vitis AI optimizer, Callbacks
1  Initialize: Training and evaluation environments,
     action noise, SAC model, and callbacks;
2  for $timestep \leftarrow 0$ to TOTAL_TIMESTEPS - 1 do
3      if $timestep ==$ PRUNING_TIMESTEP then
4          Prune the model using Vitis AI Optimizer with
             specified sparsity;
5      end
6  end
7  for $episode \leftarrow 0$ to NUM_EVAL_EPISODES - 1 do
8      Evaluate the trained model for one episode;
9      Log episode results (rewards, episode lengths);
10 end
```

## V. Evaluation

We evaluate the performance of six different SAC models trained on the MuJoCo Humanoid-v4 environment on an NVIDIA RTX 3090 GPU by running each policy on 100 unique evaluation environments to ensure robustness of results. Our trained models are in PyTorch, and we use 100 unique instances of the MuJoCo Humanoid for each model as our evaluation environments.

The models under evaluation are as follows:

- Baseline (no pruning)
- Policy network pruned after 10% training timesteps
- Policy network pruned after 20% training timesteps
- Policy network pruned after 50% training timesteps
- Policy network pruned after 75% training timesteps
- Policy network pruned after 100% training timesteps, i.e after the model has trained to completion.

TABLE I: Summary of training and inference performance for humanoid environment

| Model | Training time (seconds) | Evaluation time (seconds) | Mean cumulative reward | Mean episode length |
|---|---|---|---|---|
| SAC baseline (without pruning) | 890.60 | 11.78 | 644.99 | 131.07 |
| SAC pruned after 10000 (10%) timesteps | 1018.86 | 13.28 | 646.67 | 128.85 |
| SAC pruned after 20000 (20%) timesteps | 1038.79 | 14.19 | 781.34 | 139.47 |
| **SAC pruned after 50000 (50%) timesteps** | **978.01** | **16.72** | **812.62** | **159.68** |
| SAC pruned after 75000 (75%) timesteps | 946.83 | 13.75 | 662.24 | 131.31 |
| SAC pruned after 100000 timesteps/full training | 894.58 | 14.18 | 696.99 | 134.54 |

Mean reward and mean episode length: higher is better
Training and evaluation time measured on NVIDIA RTX 3090 GPU.

Model performance is measured by the *mean cumulative reward* and the *mean episode length*. Better policy evaluation is indicated by larger mean rewards and longer episode lengths as the agent can stay in a 'healthy state' for an extended period, where it can remain in motion without falling over. Figure 4 shows the accumulated reward evolution over time during the training process for the baselines compared to models pruned at a specific timestep.

These trained policies are each run on 100 evaluation environment, with the values for mean accumulated rewards and mean episode lengths reported in Table I. Note that the training times increase due to the pruning process itself requiring some overhead, and the evaluation time increases with improved performance as the agent under optimal policy now spends more time in the environment and accumulates higher rewards. Figures 5a and 5b show the distributions of rewards and episode lengths of the 100 evaluation instances.

In particular, pruning the model halfway through the training process leads to a model with the best evaluation performance overall, suggesting a trade-off point where the pre-trained, non-pruned model after pruning can generalize quickly

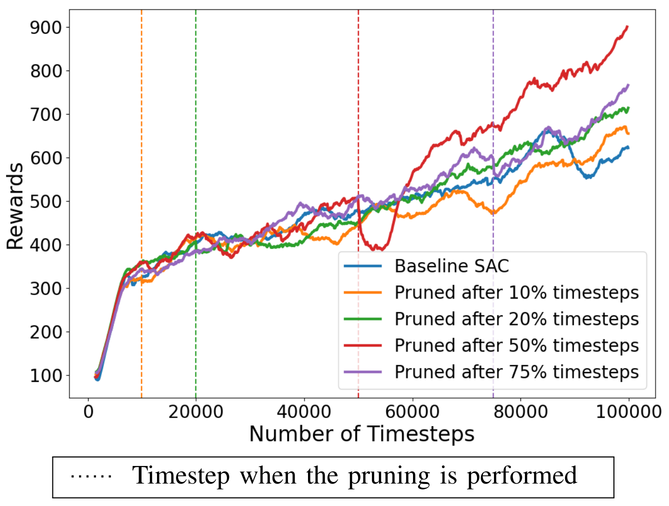


Fig. 4: Training reward evolution over time

with sufficient further training.

Overall, the results indicate that model pruning generally improves the evaluation performance of SAC models in the MuJoCo Humanoid environment and that there exists an optimal pruning timestep where performance gains can be maximized.

## VI. Future Work

We plan to extend our investigation into the other steps of the Vitis AI toolchain, specifically the quantizer and the compiler. We want to apply the complete Vitis AI workflow to several off-policy, online deep reinforcement learning (DRL) algorithms and observe the following:

- *Model Optimization*: We plan to investigate more advanced pruning techniques offered by the Vitis AI Optimizer, like Iterative Pruning and Once-for-All (OFA) pruning, which are more data-aware and require iterative refinement.
- *Model Quantization*: We plan to investigate the different quantization techniques offered by the Vitis AI Quantizer, calibrating them to obtain minimum precision while maintaining model performance.
- *Model Compilation and Deployment*: Pruned and quantized models need to be compiled and deployed to the Deep Learning Processing Units (DPUs) to fully leverage the acceleration capabilities provided by Vitis AI. This will need careful mapping to the target hardware and thorough investigation into the Xilinx runtime library (XRT).
- *Evaluation on Hardware*: We plan to implement the policy evaluation using the deep-learning processor unit (DPU) and compare the performance of DPUs with our existing results.
- *Extension to Other DRL Algorithms*: While our study focused on SAC, there exist several other state-of-the-art algorithms that should be deployed and evaluated for generalization to different classes of learning tasks, including Proximal Policy Optimization (PPO) [22], Advantage Actor Critic (A2C) [36], and Twin-Delayed DDPG (TD3) [37].

## VII. Conclusion

In this study, we explore the impact of model pruning on the performance of Soft Actor-Critic (SAC) algorithms trained on the MuJoCo Humanoid environment. We employ the Sparse Pruner Optimizer from the Vitis AI toolchain at various stages of the training process to investigate the effect of pruning on accumulated rewards and episode lengths and evaluate these models on 100 unique evaluation environments each to ensure robustness. All models were trained and evaluated on an Nvidia GTX 3090 GPU. Our findings indicate that pruning generally enhances the performance of SAC models. Specifically, pruning at 50% of the total timesteps resulted in a 23% increase in accumulated rewards and 20% increase in episode lengths, which is the maximum observed performance improvement, suggesting that this interval may strike an favorable balance between sufficient training and effective pruning. However, it is important to note that this observation is based on empirical evidence from a limited set of pruning intervals.

Through this study, we aim to provide a foundation for further research into the Vitis AI toolchain for deep reinforcement learning inference. In future work, we plan to incorporate other steps in the toolchain: model quantization, compilation and deployment to FPGA hardware for inference, and compare performance to traditional CPU and GPU implementations. Furthermore, we plan to include other state-of-the-art deep RL algorithms in our analysis, such as Proximal Policy Optimization (PPO), Advantage Actor Critic (A2C) and Twin-Delayed DDPG (TD3).

## Acknowledgement

The work was supported in part by NSF I/UCRC CNS-1822080 via the NSF Center for Space, High-performance, and Resilient Computing (SHREC).

## References

[1] A. R. Fayjie, S. Hossain, D. Oualid, and D.-J. Lee, "Driverless Car: Autonomous Driving Using Deep Reinforcement Learning in Urban Environment," in *2018 15th International Conference on Ubiquitous Robots (UR)*, 2018, pp. 896–901.

[2] V. Uc-Cetina, N. Navarro-Guerrero, A. Martin-Gonzalez, C. Weber, and S. Wermter, "Survey on reinforcement learning for language processing," *Artificial Intelligence Review*, vol. 56, no. 2, pp. 1543–1575, Feb 2023. [Online]. Available: https://doi.org/10.1007/s10462-022-10205-5

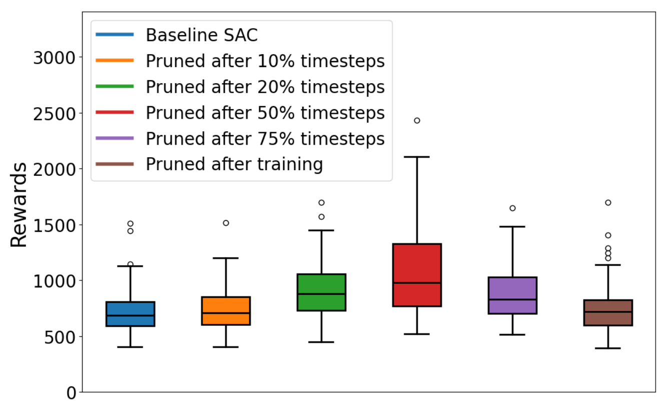
[3] S. Gadgil, Y. Xin, and C. Xu, "Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning."

[4] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks," *J. Mach. Learn. Res.*, vol. 22, no. 1, jan 2021.

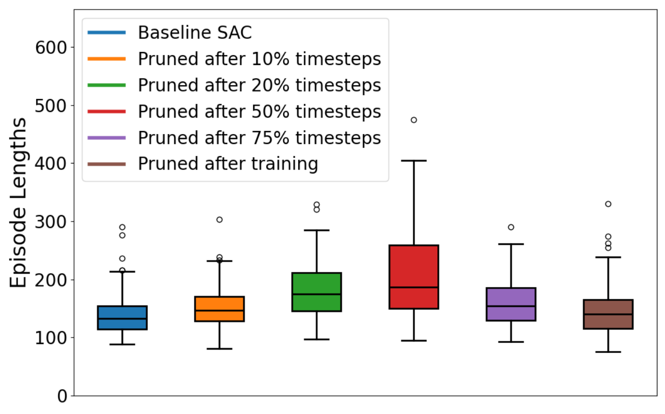
[5] A. Chaturvedi, G. Cao, and W. chun Feng, "Optimizing Deep Learning for Biomedical Imagin," in *International Conference on Computational Advances in Bio and medical Sciences*, December 2023.

[6] J. Obando-Ceron, A. Courville, and P. S. Castro, "In value-based deep reinforcement learning, a pruned network is a good network," 2024. [Online]. Available: https://arxiv.org/abs/2402.12479

[7] (2024) Vitis AI. Advanced Mirco Devices (AMD). [Online]. Available: https://github.com/Xilinx/Vitis-AI



(a) Accumulated rewards



(b) Episode lengths

Fig. 5: Policy evaluations for networks pruned at different timesteps

[8] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[9] Google-Deepmind. (2024) MuJoCo. [Online]. Available: https://github.com/google-deepmind/mujoco

[10] M. Wen, J. Kuba, R. Lin, W. Zhang, Y. Wen, J. Wang, and Y. Yang, "Multi-Agent Reinforcement Learning is a Sequence Modeling Problem," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 16 509–16 521. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/69413f87e5a34897cd010ca698097d0a-Paper-Conference.pdf

[11] S. Shao, J. Tsai, M. Mysior, W. Luk, T. Chau, A. Warren, and B. Jeppesen, "Towards Hardware Accelerated Reinforcement Learning for Application-Specific Robotic Control," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–8.

[12] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning," in *Conference on Robot Learning*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:53084610

[13] A. Ushiroyama, M. Watanabe, N. Watanabe, and A. Nagoya, "Convolutional neural network implementations using vitis ai," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0365–0371.

[14] A. M. Cabrera, Y. A. Yucesan, F. Y. Liu, W. Blokland, and J. S. Vetter, "Errant Beam Detection Using the AMD Versal ACAP and Vitis AI," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, 2023, pp. 1–6.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[17] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." in *ICML*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 1856–1865. [Online]. Available: http://dblp.uni-trier.de/db/conf/icml/icml2018.html#HaarnojaZAL18

[19] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *ArXiv*, vol. abs/2005.01643, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218486979

[20] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[21] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International Conference on Machine Learning*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:54457299

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms." *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15

[24] S. Dankwa and W. Zheng, "Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent," in *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, ser. ICVISP 2019. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3387168.3387199

[25] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016, cite arxiv:1606.01540. [Online]. Available: http://arxiv.org/abs/1606.01540

[27] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, A. Hogan, lorenzomammana, yxNONG, AlexWang1900, L. Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, F. Ingham, Frederik, Guilhen, Hatovix, J. Poznanski, J. Fang, L. Y. , changyu98, M. Wang, N. Gupta, O. Akhtar, PetrDvoracek, and P. Rai, "ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements," Oct. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4154370

[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[29] A. Ushiroyama, M. Watanabe, N. Watanabe, and A. Nagoya, "Convolutional neural network implementations using vitis ai," in *2022 IEEE*

*12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0365–0371.

[30] J. Wang and S. Gu, "Fpga implementation of object detection accelerator based on vitis-ai," in *2021 11th International Conference on Information Science and Technology (ICIST)*, 2021, pp. 571–577.

[31] M. Machura, M. Danilowicz, and T. Kryjak, "Embedded object detection with custom littlenet, finn and vitis ai dcnn accelerators," *Journal of Low Power Electronics and Applications*, vol. 12, no. 2, 2022. [Online]. Available: https://www.mdpi.com/2079-9268/12/2/30

[32] Y. Fukuda, K. Yoshida, and T. Fujino, "Evaluation of model quantization method on vitis-ai for mitigating adversarial examples," *IEEE Access*, vol. 11, pp. 87 200–87 209, 2023.

[33] A. M. Cabrera, Y. A. Yucesan, F. Y. Liu, W. Blokland, and J. S. Vetter, "Errant beam detection using the amd versal acap and vitis ai," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, 2023, pp. 1–6.

[34] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4906–4913.

[35] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[36] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 1928–1937.

[37] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.