An Energy-Efficient Hardware Accelerator for Hierarchical Deep Reinforcement Learning

Aidin Shiri†, Bharat Prakash†, Arnab Neelim Mazumder†, Nicholas R. Waytowich‡, Tim Oates†, Tinoosh Mohsenin†
†Department of Computer Science & Electrical Engineering, University of Maryland Baltimore County
‡US Army Research Laboratory

***Abstract*—Reinforcement Learning (RL) has shown great performance in solving sequential decision-making and control in dynamic environments problems. Despite its achievements, training Deep Neural Network (DNN) based RL is expensive in terms of time and power because of the large number of episodes required to train agents with high dimensional image representations. At the deployment also, the massive energy footprint of deep neural networks can be a major drawback. Embedded devices as the main deployment platform, are intrinsically resource-constrained and deploying DNN on them is challenging. Consequently, reducing the number of actions taken by the RL agent to learn desired policy, along with the development of efficient hardware architectures for RL is crucial. In this paper, we propose a novel hardware architecture for RL agents based on the learning hierarchical policies method. We show that hierarchical learning with several levels of control improves RL agents training efficiency and the agent converges faster compared to a none hierarchical model and therefore using less power. This is especially true as the environment becomes more complex with multiple objective sub-goals. Our method is important for efficient learning of policies for RL agent, especially when the target platform is a resource constraint embedded device. By performing a systematic neural network architecture search and hardware design space exploration, we implemented an energy-efficient scalable hardware accelerator for the hierarchical RL. Hardware factors of merit such as the latency, throughput, and energy consumption of the accelerator are evaluated with the various processing elements, and model parameters. The most energy-efficient configuration achieves 139 fps throughput with 5.8 mJ energy consumption per classification on Xilinx Artix-7 FPGA. Compared to similar works our design shows up to 3x better energy efficiency.**

***Index Terms*—Reinforcement Learning, Energy Efficient Hardware, FPGA, ASIC**

I. Introduction

Reinforcement Learning (RL) is a goal-oriented paradigm of machine learning in which the agent tries to learn a policy to achieve complex tasks by trial and error. Reinforcement Learning is used for problems that involve sequential decision making where the agent needs to take actions in an environment to maximize cumulative future rewards. In reinforcement learning, goals are specified using a reward function [1]. Human feedback can also be used to specify goals as shown in [2]. Despite great interest in hardware acceleration of deep neural network based AI applications in recent years, most of the previous works in Reinforcement Learning focused on the algorithm and theoretical aspects, and very few works have considered hardware implementation for RL [3], [4], [5], [6].

Learning the policy for the RL agent can be a complex and time-consuming task. It is important to train the agent in the fastest and most efficient way, especially when the agents act in the real-world environment where performing actions could be expensive. One scalable way to do this is by using a hierarchical reinforcement learning agent. In this method intended policy is divided into individual sub-policies. Two neural networks are trained for performing the desired task, one for selecting the proper subgoal, and the other for producing the proper action to perform in the environment. Besides making it easy to specify goals and rewards, hierarchical subgoals can also be reused in other similar environments that might have the same subgoals.

In this paper, we propose a hardware friendly architecture for hierarchical reinforcement learning which can learn the desired policy faster than baseline and act in real-world scenarios. This paper makes the following contributions:

- Propose a hardware-friendly architecture for Reinforcement Learning which works based on hierarchical policy learning.
- Perform a Neural Network Architecture Search to find the best trade-off between accuracy (task completion), model size, and computation.
- Propose a scalable and parameterized hardware in Verilog HDL for deploying on the embedded devices.
- Implement the proposed work with different configurations in terms of processing elements and parallelism on Artix7 FPGA and compare them in terms of power consumption, energy efficiency, latency, and utilized resources.

II. Proposed System Architecture

A. Background

In Reinforcement Learning, the agent is trained to perform a certain task which usually requires performing a sequence of decisions without a supervisor. The agent only gets a reward based on the completion of the tasks to learn the desired policy without specifying how to accomplish the task. Generally, Reinforcement Learning is modeled through Markov Decision Processes (MDP). MDP is a list of elements (S, A, P, R, γ), which denotes state space, action space, transition function, reward function, and discount factor respectively. The agent tries to interact with the environment through a set of possible

moves called actions and the environment takes the agent's action and current state and returns a reward and next state. The reward is feedback from the environment by evaluating the agents performance for completing the task. The policy is the strategy with which the agent maps its state to the action that guarantees the optimum future reward. The trajectory is a sequence of state actions.

B. Hierarchical Reinforcement Learning

Reinforcement learning techniques have shown great promise in the past few years, getting us a few steps closer to deploying them in the real world. However, there are still many challenges before we can safely and efficiently deploy them in the real world, especially when these technologies work closely with other humans. One of the issues is low sample efficiency, where we require a large number of resources in terms of computing time and data to train these autonomous agents. One way to tackle this problem is to make use of the hierarchical structure present in complex sparse feedback tasks to improve learning. It is possible to use the hierarchical structure and learn multiple levels of control to solve tasks as shown by [7]. These methods have shown to improve the performance in sparse reward and long-horizon tasks. In this work, we use these architectures with 2 levels of control and also with the assumption that we know the subgoals to solve the task. We then show hardware implementations for this architecture. The following sections show the architecture and RL training details.

C. RL environment and training the RL agent

Our experiments are performed on the MiniWorld environment [8]. It is a minimalistic 3D environment for Reinforcement Learning and robotics research. The agent can navigate rooms and manipulate objects using its first-person view as shown in Fig. 2 In our experiments we set up a simple scenario where the agent is in a room that has a box and a key. The agent is spawned at random locations in each episode. The task is to first collect the red key and then reach the green box. The agent receives a reward when it reaches the box, provided it has collected the key first. If the agent reaches the box without the key, it receives no reward. As described earlier this task can be decomposed into 2 subgoals: 1. reach the key, 2. reach the box. The architecture is shown in Fig. 1 where the agent is trained in 2 phases. First, the low-level policy (πC) is trained to perform the 2 subgoals. This is done by concatenating a one-hot encoded task id to the actor network. In the second phase, the high-level policy (πG) is trained to select subgoals. This output is then concatenated to the low-level policy, which executes a policy to reach the subgoals. Both policies are trained using proximal policy optimization [9]. The high-level policy does not need to output a goal at every environment time step. It outputs a subgoal and sleeps for several time steps (K), which is then used by πC to output low-level actions. This also helps to reduce the hardware and time complexity. We experiment with different intervals, K as shown in the results section in Fig 3.

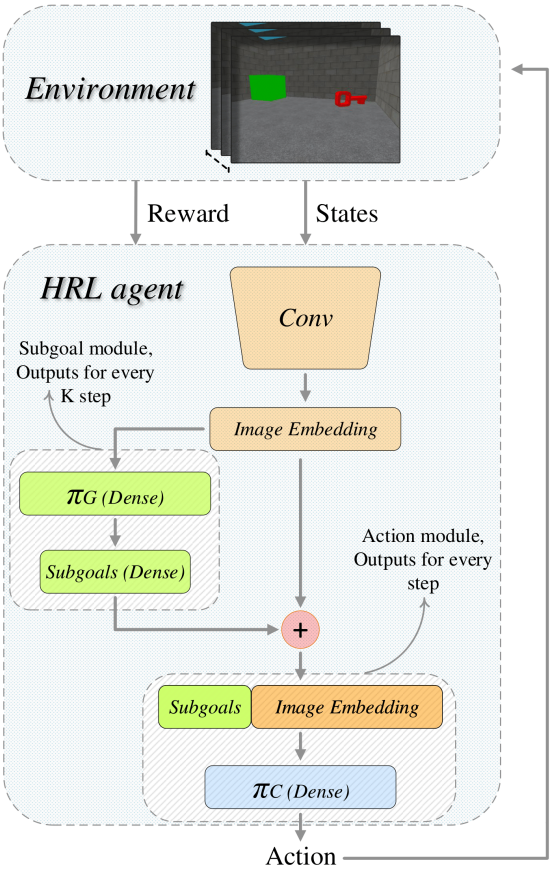

Fig. 1: The proposed architecture of the hierarchical RL agent

TABLE I: Initial HRL neural network architecture in details

Layer	Input	Window	# Kernels	Activation
Conv 1	80*60*3	(5,5)	32	ReLU
Conv 2	38*28*32	(5,5)	32	ReLU
Conv 3	17*12*32	(4,4)	32	ReLU
Flatten	7*5*32	-	-	-
Dense [πG]	64	-	-	ReLU
Dense [πC]	64	-	-	ReLU
Action	3	-	-	Softmax

III. Neural Network Architecture Search and Experimental Results

Neural Architecture Search (NAS), is the process of systematic neural network architecture engineering, to achieve the best possible performance with the smallest model size and least computation. We performed the NAS on the model search space and evaluate our model performance. Search space defines the networks that can be examined to produce the final architecture. The initial model consists of three convolution layers for generating image embeddings. This embedding is fed to a dense (fully-connected) layer to produce the subgoal vector. Afterward, subgoal vector and image embedding are concatenated together to fed another dense layer with Softmax activation and generate the desired output. The

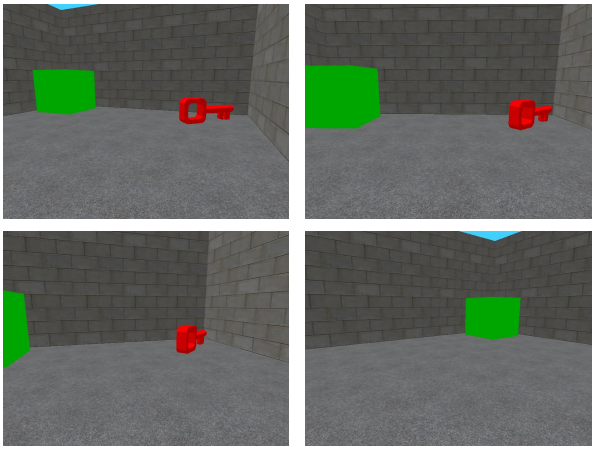

Fig. 2: The Miniworld environment in which the agent tries to learn the policy of picking up an instructed object. The environment back end is modified so that the agent could be instructed.

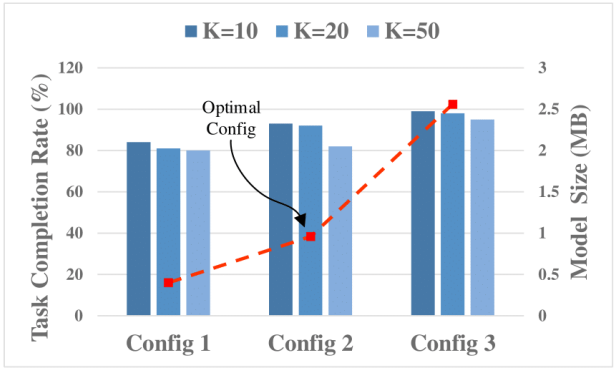

Fig. 3: Performance and Memory size of hierarchical RL agent for different configurations with respect to K

initial HRL agent architecture is explained in detail in III. We performed a macro-architecture search for layer type, hyper-parameters, and connections with other layers to achieve the best performance. Furthermore, for each block, we optimize the convolutional layer kernel size and filter number, as well as the dropout rate, the existence of a pooling layer after each convolution layer. Three of the best hardware architectures that produced the best results are summarized in Table II. By comparing the number of parameters, and required memory space for each configuration, we observed that Config 2 achieves almost the same reward as the best configuration, Config 3, while requiring less than 2 times memory size. Therefore, we select Config 2 as the optimal design for implementing on FPGA hardware. Figure 3 illustrates the model size and performance of different configurations with respect to the time step, K, amount of time steps that πG sleeps. We observe that increasing K, degrades the model performance.

TABLE II: Four network configurations with different computation and memory complexity. Each model is trained for one million Steps.

	RL	HRL		
		Config 1	Config 2	Config 3
# Parameters	117K	53K	**126 K**	335K
Memory (MB)	0.45 MB	0.4 MB	**0.96 MB**	2.56 MB
Task Completion	55%	84%	**93%**	98%

IV. FPGA Implementation Results and Design Space Exploration

Figure 4 shows the detailed block diagram of the proposed hardware architecture. We designed and implemented our hardware on low-power Xilinx Artix-7 200t FPGA. Hardware is designed to be configurable with different hyper-parameters to meet custom application requirements such as low power consumption or high throughput. Parameters include the number of processing elements (PE), filter shapes, number of filters in the convolutional layers, sizes of the dense layers that are configurable to engage maximum parallel processing ability or utilize the least possible hardware resources. The hardware consists of three main units: (1) convolution block for performing convolution operation. (2) fully-connected block that implements the dense layer functionality (3) on-chip memory which stores image embedding and subgoal vector. The convolution block operates with a multiplier embedded within it. Along with that, there are separate memory blocks for saving weights and feature maps. The input data is fed to the convolution block to calculate the $valid$ convolution of the input using a multiplication unit (MU) and an adder with ReLu activation logic. Strides of two in each direction replaced the time consuming max-pooling operation. Following the convolutional layers, the first full-connected layer reads the data from the image embedding memory and operates using one multiplier-adder, and a few registers. The address flow and control unit generates memory addresses depending on the layer functionality of either convolution or dense. Later, the subgoal vector generated by the fully-connected layer is concatenated with the output of the convolution layer, and form the input of the final fully-connected layer with Softmax activation to produce proper action. The output of this layer is the action generated by the agent to navigate toward the relevant task within the environment. The control unit manages the data flow through different parts of the hardware. 32-bit fixed-point number format is used for representing the data inside the hardware. We implemented our design with 1,2,4 and 8 PEs to measure the latency, throughput, power consumption, and energy efficiency of each case. Table III shows the hardware implementation results for each case at three different frequencies.

V. Conclusion

In this paper, we proposed an energy-efficient hardware architecture for hierarchical reinforcement learning. We ver-

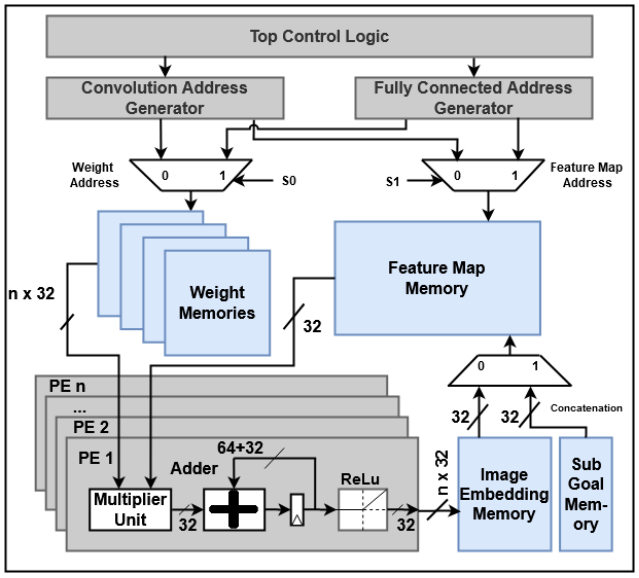

Fig. 4: Block diagram of hardware architecture which consists of feature map memory and weight memory that are accessed by the convolution and fully-connected address control unit to fetch data for the Processing Elements (PEs).

TABLE III: FPGA implementation result of the best configuration with different number of processing elements (PE) on Artix7-200t

	This Work				Existing Works	
	1 PE	2 PE	4 PE	8 PE	[10]	[11]
Application	Reinforcement Learning				Image Classification	
Frequency (MHz)	100	100	100	100	100	150
LUT	1876	2073	2428	3345	NR	NR
BRAM	321	321	321	321	NR	545
DSP Slice	11	15	23	39	NR	391
Power (mW)	**445**	569	811	873	1015	10000
Latency (mS)	30.1	15.1	7.2	3.9	16	38.1
Throughput	33	66	139	**256**	63.5	26.2
Energy (mJ)	13.4	8.6	**5.8**	33.4	33.7	381
Performance (GOPS)	0.6	1.1	2.4	**4.4**	1.67	38.4
Energy Efficiency (GOPS/W)	1.3	1.9	3	**5**	1.65	3.84

ified our design by testing its accuracy in the Miniworld environment. Performance of the model is measured with three different configurations and the best configuration is selected to implement on hardware. The proposed hardware architecture is scalable with different number of PEs and could be configured to achieve high throughput or low power consumption. It achieves 256 frames per second throughput with 33 mJ energy consumption per classification when configured to run at maximum speed. At low power configuration, the hardware consumes less than 450 mW for 33 frames per second throughput. Comparing to the related works that use similar case studies and platform, our hardware achieves up to 3x better energy-efficiency and 4x better throughput.

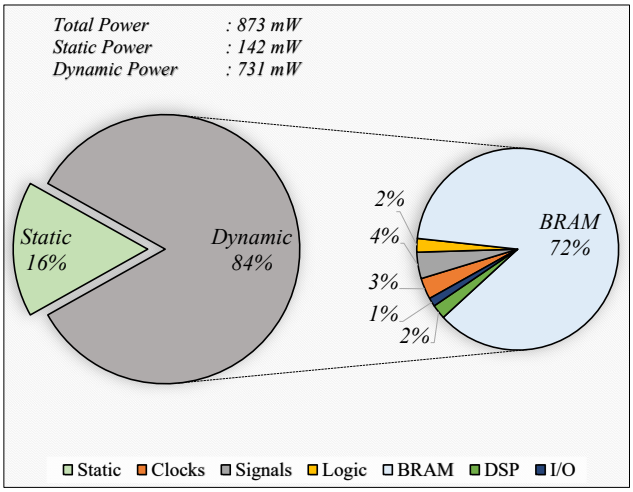

Fig. 5: Power consumption breakdown of HRL accelerator on Artix7 FPGA

VI. Acknowledgment

This project was sponsored by the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF-10-2-0022.

References

[1] R. S. Sutton *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2, no. 4.

[2] S. Gandhi *et al.*, "Learning behaviors from a single video demonstration using human feedback," 2019.

[3] A. Shiri *et al.*, "Energy-efficient hardware for language guided reinforcement learning," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 131–136.

[4] A. Shiri *et al.*, "A hardware accelerator for language guided reinforcement learning," *IEEE Design & Test*, 2021.

[5] N. K. Manjunath *et al.*, "An energy efficient edgeai autoencoder accelerator for reinforcement learning," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 182–195, 2021.

[6] B. Prakash *et al.*, "On the use of deep autoencoders for efficient embedded reinforcement learning," in *ACM Proceedings of the 29th Edition of the Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 2019.

[7] P.-L. Bacon *et al.*, "The option-critic architecture," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[8] M. Chevalier-Boisvert, "gym-miniworld environment for openai gym," https://github.com/maximecb/gym-miniworld, 2018.

[9] J. Schulman *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[10] G. Zhong *et al.*, "Synergy: An hw/sw framework for high throughput cnns on embedded heterogeneous soc," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 2, pp. 1–23, 2019.

[11] S. Moini *et al.*, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 10, pp. 1217–1221, 2017.

[12] M. Hosseini *et al.*, "A fast method to fine-tune neural networks for the least energy consumption on fpgas," in *Proceedings of the Hardware Aware Efficient Training workshop of ICLR 2021*, 2021.

[13] A. N. Mazumder *et al.*, "Automatic detection of respiratory symptoms using a low power multimodal cnn processor."

[14] H. Ren *et al.*, "End-to-end scalable and low power multi-modal cnn for respiratory-related symptoms detection," 2020.

[15] B. Prakash *et al.*, "Improving safety in reinforcement learning using model-based architectures and human intervention," *arXiv preprint arXiv:1903.09328*, 2019.