A Hardware Accelerator for Language-Guided Reinforcement Learning

Aidin Shiri, Arnab Neelim Mazumder, and Bharat Prakash
Department of Computer Science and Electrical Engineering, University of Maryland,
Baltimore County, Baltimore, MD 21250 USA

Houman Homayoun
University of California at Davis, Davis, CA 95616 USA

Nicholas R. Waytowich
U.S. Army Research Laboratory, Adelphi,
MD 21005 USA

Tinoosh Mohsenin
Department of Computer Science and Electrical Engineering University of Maryland,
Baltimore County, Baltimore, MD 21250 USA

Editor's notes:
Reinforcement learning guided with language instructions can potentially improve efficiency and better align with overall system design goals. This article introduces a hardware-friendly architecture for training reinforcement learning equipped with natural language instructions.
—Sai Manoj, George Mason University

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. RL 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 RL goals are specified using a reward function [1]. Human feedback can also be used to specify goals as shown in [2] and [3]. It is important to train the agent in a way that easily abides by human instructions. One scalable way to do this is by using language instructions. Recent work in using language to guide RL agents has gained a great interest among AI researchers [4], [5]. These methods have shown improvement in making agents able to understand human language. Instructions are processed using language processing techniques to generate embeddings to feed the agent along with the states. Besides making it easy to specify goals and rewards, language can also be used to convey constraints and improve the safety of RL agents [6]. Although hardware implementation of deep neural networks has gained lot of attention in recent years [7]–[9], most of the previous works in RL focused on the algorithm and theoretical aspects, and very few works have considered hardware implementation for RL [10]. In this article, we propose a hardware-friendly architecture for RL which can interpret language instructions to act in real-world scenarios. This article makes the following contributions:

- Propose a hardware-friendly architecture for RL which can interpret structured language instructions.
- Evaluate the model learning efficiency using two real-world environments with different levels of complexity.
- Propose scalable, parallel, and parameterized hardware in Verilog HDL that can receive image inputs from environment and language constraints for deploying on the embedded devices.

Digital Object Identifier 10.1109/MDAT.2021.3063363
Date of publication: 2 March 2021; date of current version: 22 April 2022.

- Implementation of the proposed work with different configurations of processing elements (PEs) on low-power Artix-7 FPGA and providing a detailed analysis of energy efficiency of the design.
- Post-layout ASIC implementation of the best case hardware on 14-nm FinFET and providing results and analysis in terms of power consumption, throughput, and area utilization.

Proposed method

Background

In RL, 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, RL 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. In this article, we propose an architecture similar to the conventional RL structure described above; however, by adding a language module to the design we make the agent capable of performing the instructed task. In other words, the agent learns the policy of performing the desired task instructed by the language module.

Proposed system architecture

In the conventional RL model, agents interact with the environment through a set of actions and try to learn a certain policy by gaining the maximum possible reward. In this work, we also added a language module that gives certain structured instructions to the agent to accomplish the desired tasks. Figure 1 illustrates the architecture of RL with the addition of an "Instructions" module during inference. The proposed system architecture consists of two main parts: the traditional RL agent and the language module. The language module consists of an embedding and a gated recurrent unit (GRU) which translates text instructions to a vector (not shown in the figure). The output of the language module is concatenated with the output of the agent and makes one vector pass through fully connected layers. The RL agent receives an image and an instruction in the form of text and generates action at each step. The image is processed by a convolutional neural network to output an image embedding. The language input is processed by an embedding layer followed by a GRU unit to get a text embedding.

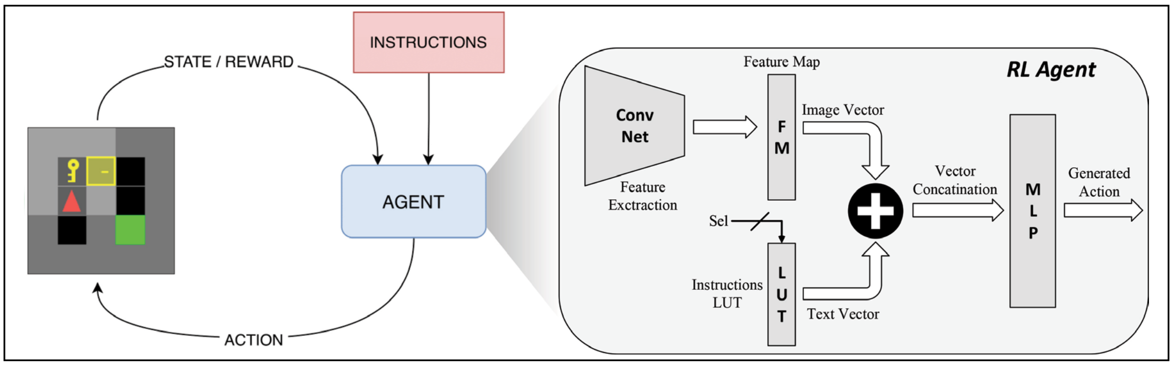

Figure 1. Proposed architecture of the RL agent with language instructions for interacting with the environment while getting structured language constraints. In the conventional reinforcement learning model, agents interact with the environment through a set of actions and try to learn a certain policy by gaining the maximum possible reward. Meanwhile, adding the language instructions module force the agent to learn to accomplish the dictated goal. The networks receives the states as images, along with instruction.

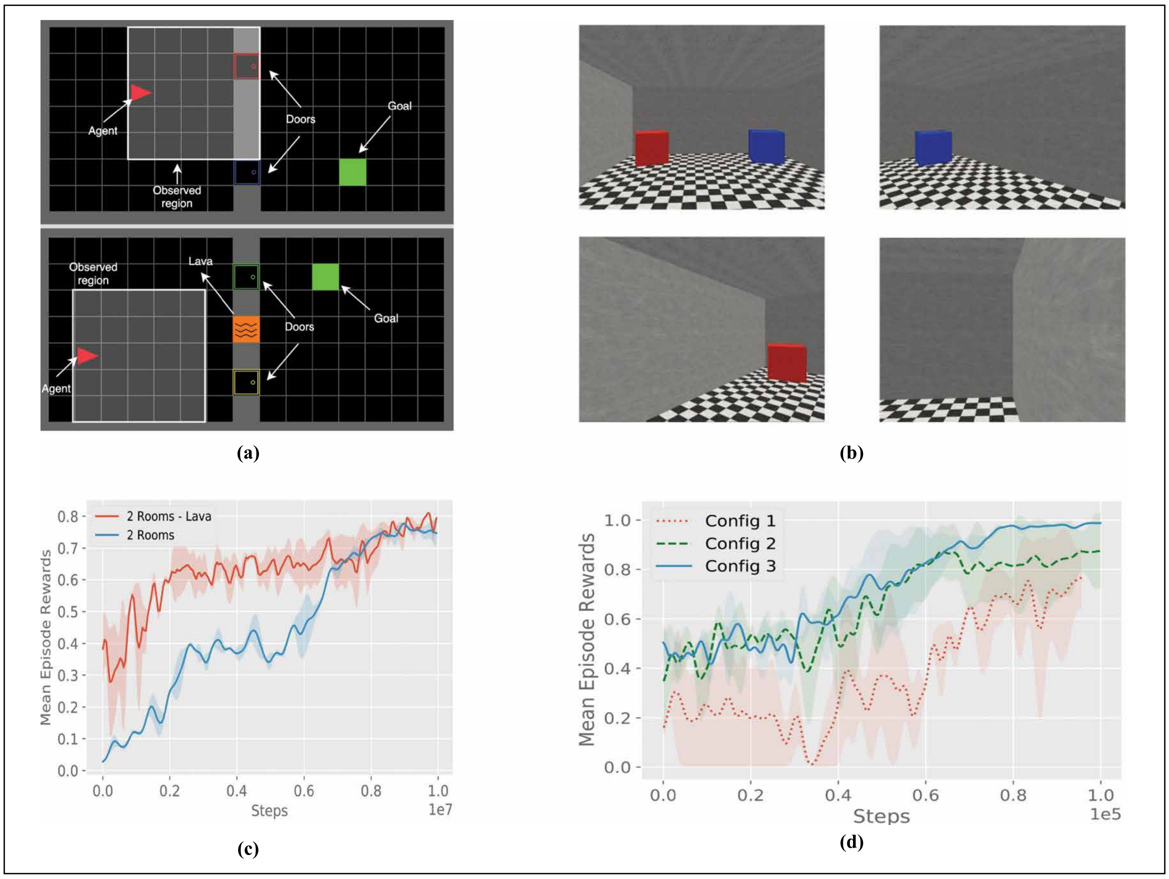

Figure 2. Two real-world environments with different complexities that are used to evaluate the training performance of our model. (a) MiniGrid environment, two rooms, and two room-lava. (b) Miniworld environment, four different views from agent perspective. (c) MiniGrid mMean episode rewards. (d) Miniworld mean episode rewards for three different network configurations.

These two embeddings are then concatenated and passed through an MLP to get the final action. The model is trained using proximal policy optimization. During the inference, the bulky language module is replaced with a simple lookup table (LUT) that can feed the saved instruction vectors to the agent.

Experimental results

In this section, we explain the experimental setup for testing the RL agent performance for learning the instructed policy. Two environments with different levels of complexity are selected to evaluate the RL agent performance with the proposed configuration. Gym MiniGrid [11] and Miniworld object pickup [12] environments are selected since one has less complexity and the other is more complex in terms of details in the environment and the instructed policy of the RL agent. For each case, the network is trained separately with feedback from the environment.

Case study one: MiniGrid

This environment has multiple rooms with doors, walls, and goal objects. The doors can have multiple colors and the agent and goal objects are spawned at random locations. The action space is discrete which allows movement in all four directions, opening and closing doors and picking up and dropping objects. The environment is partially observable, the agent can only see an ego-centric 5×5 view in front of the agent. Also, the agent cannot see through walls and closed doors. We designed multiple scenarios in this environment with increasing difficulty levels both in terms of the tasks and the language instructions. We compare the performance of our model with a

Table 1. Example language constraints used in the different environments to instruct specified policy to the agent.

#	Instructions	Environments
1	do not use the red door	MiniGrid, 2Rooms
2	do not go through the blue door	MiniGrid, 2Rooms
3	no yellow door	MiniGrid, Lava
4	do not go through the red door and stay away from lava	MiniGrid, Lava
5	avoid blue door and no lava	MiniGrid, Lava
6	Pick up the red box	Miniworld, Pickup Objs
7	Go to the blue box	Miniworld, Pickup Objs

baseline where we shape the environment rewards directly by giving negative rewards for violations.

The two-room scenario is shown in Figure 2a (top). It consists of two rooms separated by a wall and two doors. In each episode, the agent and the goal are spawned at random locations and the door colors are randomly initiated. Also, there is random language instruction generated which follows a fixed grammar as shown in the examples in Table 1. The task is to reach the green goal object using the minimum number of steps while also going through the correct door (which is specified using the language instruction).

The two rooms with lava scenario are shown in Figure 2a (bottom). This is similar to the two rooms environment but it has an additional cell called the lava. It is the orange cell seen in the figure and it can behave in one of two ways depending on the language instruction. Example language instructions are given in Table 1. For the instruction of types 1–3, the lava acts as a teleportation cell, the agent entering the lava will be instantly moved right next to the agent. If the instruction is of types 4–5, the agent entering the lava will die and result in zero reward.

Case study two: Miniworld

To observe the performance of the model in a more complex environment, we evaluated its performance in the MiniWorld environment as well. The second set of experiments are performed on the modified MiniWorld environment. MiniWorld is a minimalistic 3-D environment for RL and robotics research. The agent can navigate rooms and manipulate objects using its first-person view as shown in Figure 2b. In our experiments, we set up a simple scenario where the agent is in a room and there are two boxes, one red and one blue box. The agent is spawned at random locations at each episode and it receives instructions in the form of text as given in Table 1. Depending on the instruction, the agent has to reach one of the boxes. The agent gets a reward of 1 if it reaches the instructed box to pick, and 0 otherwise. The neural network architecture of the agent for this environment consists of three convolutional layers, followed by two fully connected and an MLP layer. Images captured from the environment are passed into the convolutional layers and important features are extracted from images using a set of local filters. Once the convolution operation is performed, the image gets flattened into a single vector. Stride of 2×2 is used after each convolution layer instead of a max-pooling operation, due to its simpler hardware implementation. During the test, the language module discussed in the previous section is replaced with a simple LUT which keeps the known structured instructions. The output vector of convolution layers is concatenated with the desired instruction vector from instructions LUT and fed into two fully connected layers followed by an MLP to generate the corresponding action.

One of the main goals of this article is to propose efficient embedded hardware for RL that can meet the throughput, area, and power budget of embedded devices. We tested our architecture with three different configurations for the RL agent to observe the performance of the model with respect to the number of computations and parameters. In deep neural networks, convolutional layers have the majority of the computation load of the architecture. To tackle the intensive computation problem, we reduce the number of filters and evaluate the design with three different filter sets for the convolution layer. The RL agent is trained using the proposed architecture and its performance is measured by recording the reward of completing the instructed task.

Table 2. Three network configurations with different kernel sizes. The baseline architecture is Config 3.

	Config 1	Config 2	Config 3
# Conv Kernels	8	16	32
# Parameters	49 K	86 K	176 K
# Computations	4 M	10.9 M	33.3 M
Memory (MB)	0.39	0.69	1.41

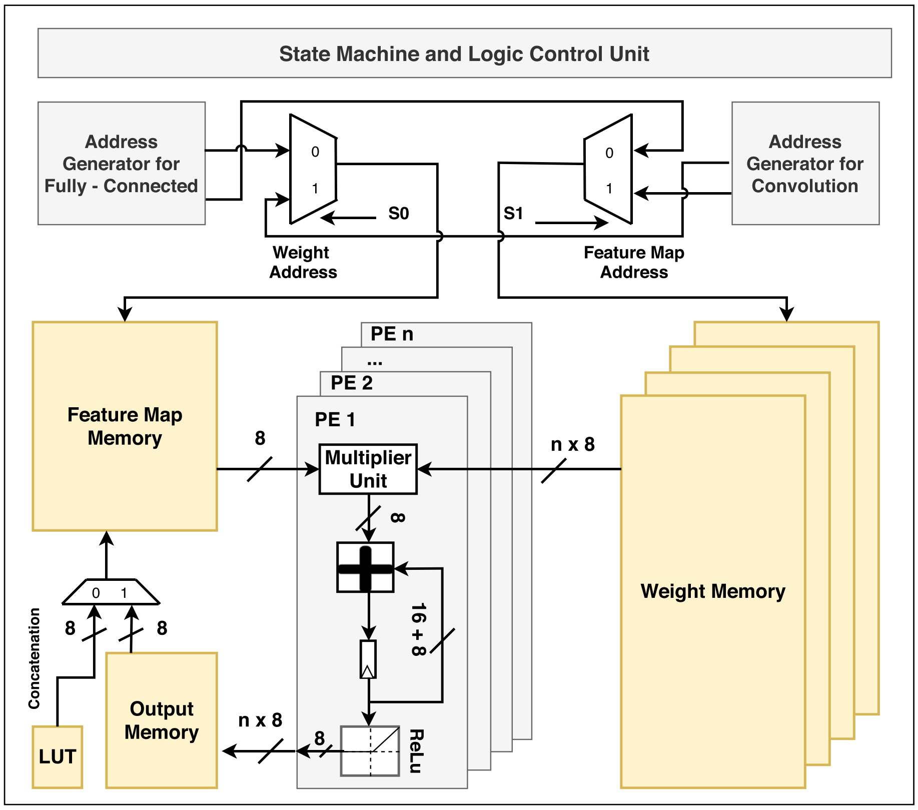

Figure 3. Block diagram of hardware architecture which consists of feature map memory and weight memory that are accessed by the convolution and fully connected address control unit to fetch data for the PEs.

The first, second, and third convolutional layers are tested with the configuration of 8, 16, and 32 filter sets, respectively. Table 2 summarizes the number of parameters, computations, and required memory space for each configuration, and Figure 2d shows the reward function of each case with respect to the number of episodes. It can be observed that Config 2 achieves almost the same reward as the best configuration, Config 3 (the baseline), while requiring 2× less memory and 3× less computation. Therefore, we select Config 2 for implementation on the hardware.

Proposed hardware accelerator

Accelerator architecture design

The block diagram of the proposed hardware architecture is illustrated in Figure 3. Hardware is designed to be configurable with different parameters to meet custom application requirements such as low power consumption or high throughput. Parameters including the number of PEs, filter shapes, number of filters in the convolutional layers, sizes of the dense layers are configurable to whether engage maximum parallel processing ability of the target platform or utilize the least possible hardware resources.

The proposed hardware integrates three core microarchitectures: 1) PEs that are responsible for the calculation of the Mac operation of the convolutional and fully connected layers with a ReLU activation function; 2) address generators which read the weights and features from the memory and provide the PEs with proper data; and 3) the memory blocks that store the instructions, weights, and features of the network. The convolution block operates with a multiplier embedded in its design. Along with that, there are separate memory blocks for storing weights and feature maps. The input data are fed to the convolution block which calculates the valid convolution of the input using a multiplication unit (MU) and an adder with ReLu

Table 3. Energy-efficiency results GoPS/W for the different number of PEs and clock speeds implementations on Artix-7 FPGA.

	25 MHz	50 MHz	90 MHz
1 PE	0.43	0.73	1.04
2 PE	0.81	1.31	1.78
4 PE	1.44	1.31	2.81
8 PE	2.84	4.26	**5.45**

Table 4. Results of post place and route ASIC implementation of the proposed hardware accelerator with 8-bit memory width and 8 PEs in 14-nm technology.

Hardware Metric	Result
Area (mm^2)	3.3
Clock Frequency (MHz)	150
Static Power (uW)	16.43
Dynamic Power (mW)	9.14
#512×32-bit SRAM Blocks	169
Performance (GOP/S)	2.4
Throughput (fps)	361.6
Energy Efficiency (GOPS/W)	262.5

activation logic. Strides of two in each direction replaced the time-consuming max-pooling operation. Following the convolutional layers, the first fully connected layer reads the data from the feature map memory and performs the operation 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. At this moment, the instruction vector stored in the LUT is concatenated with the output of the first fully connected layer and forms the input of the next fully connected layer. As discussed in previous sections, an LUT memory with a pretrained instructions vector replaces the bulky language processing module during the test. The generated vector is fed to another fully connected layer that performs the same operation and finally, the last layer, an MLP layer generates the proper outputs by the agent to navigate through the environment to perform the instructed policy. The control unit is a finite state machine that orchestrates different microarchitectures of the hardware. The 8-bit fixed-point format is used for representing the weights and features inside the accelerator to guaranty accuracy and power consumption balance.

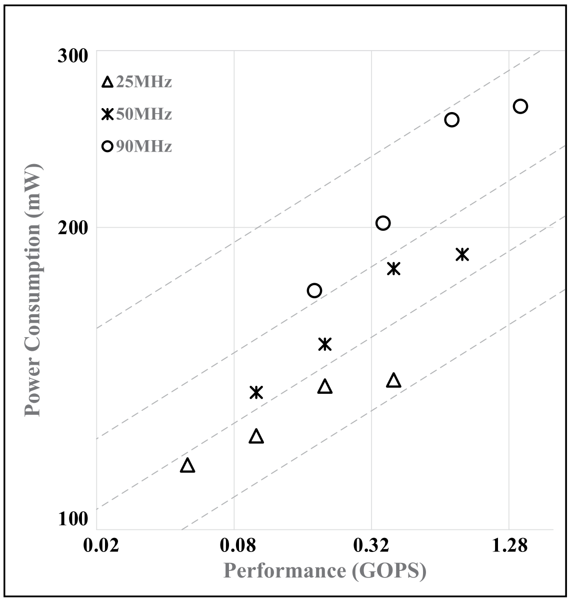

Figure 4. Scatter plot of energy efficiency tradeoff between performance and power for the FPGA implementation results. Dynamic power and performance are sketched with respect to the different frequencies and number of PE. Most energy efficient architecture-8 PE-is selected for ASIC implementation.

Hardware implementation results and analysis

We implemented the proposed architecture with 1, 2, 4, and 8 PEs on Xilinx Artix-7 FPGA to measure the latency, throughput, power consumption, and energy efficiency of each case on a low-power edge device. Detailed hardware implementation results along with its analysis are presented in Table 4 of [10]. The best configuration in terms of energy efficiency uses eight PEs running at 90-MHz frequency, which results in an average 5.45-GOPS/W energy efficiency and 1.2-mJ energy consumption per inference. This configuration consumes 264 mW to generates 217 actions per second which is much higher than the typical 30 frames per second (fps) requirement of the vision camera devices. For low-power applications, one can run this design with 4 PEs at 25 MHz to meet the 30-fps requirement while consuming less than 139 mW on an Artix-7 FPGA.

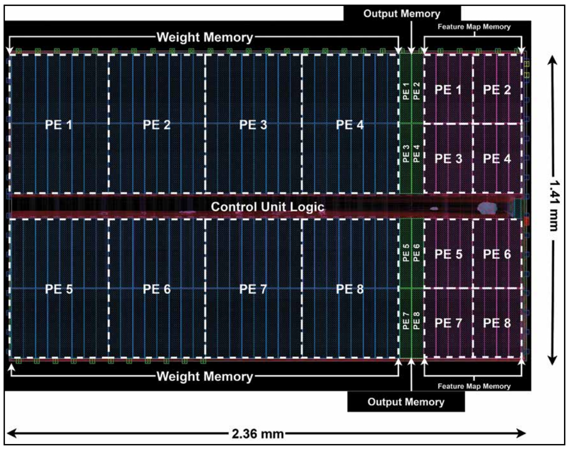

Figure 5. Post-layout view of the proposed hardware accelerator for ASIC implementation in 14-nm FinFET technology with the operating frequency of 150 MHz.

The energy efficiency of an accelerator is equal to dividing the performance (GOPS) over the power consumption. Increasing the number of PEs and frequency increases power and energy consumption, but since the increase in terms of throughput is higher than the former two, it results in better power efficiency. As illustrated in Table 3, it can be observed that while lower frequencies and the number of PEs yield lower power consumption and increasing them result in better energy efficiency. Figure 4 shows the scatter plot of hardware energy efficiency tradeoff between performance and power.

To further reduce the overall power consumption and increase the energy efficiency of the hardware accelerator, we implement the most energy-efficient architecture of the FPGA implementations, the configuration with eight number of PE, as an application-specific integrated circuit (ASIC). A standard cell register transfer level (RTL) to 14-nm FinFET post-layout with 0.8-V power supply is implemented using Synopsys Design Compiler and Integrated Circuit Compiler II. Since the ASIC implementations are more efficient than FPGA implementations, the clock frequency can easily be increased to 150 MHz to further increase the energy efficiency of the design. ASIC implementation results show 29× reduction in power consumption and 48× improvement in energy efficiency compared to the FPGA implementation results. The details of ASIC implementation are provided in Table 4 and the layout is shown in Figure 5.

In this work, we proposed an energy-efficient hardware architecture for RL that receives structured language as instructions for better guidance and training. We evaluated our model by testing it in several real-world environments with different complexities. Performance of the model is measured with different configurations and the best configuration is selected to implement on hardware. We designed a scalable hardware architecture that can be configured to achieve high throughput or low power consumption. Multiple architectures with different parameters are implemented on FPGA to find the most energy-efficient configuration and the best case is selected for ASIC implementation. Although we emphasize the energy-efficiency in the deployment of RL, there is no similar work that tackles this problem. We believe that the new wave of deploying machine learning algorithms on the edge devices, such as autonomous vehicles or robotic applications, along with the increase of open source RL environments and frameworks will result in great demand for energy-efficient embedded RL applications and facilitate future research in this direction. ■

Acknowledgments

This work was supported by the U.S. Army Research Laboratory under Cooperative under Agreement W911NF-10-2-0022. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, vol. 2, no. 4. Cambridge, MA, USA: MIT Press, 1998.

[2] P. F. Christiano et al., "Deep reinforcement learning from human preferences," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4299–4307.

[3] S. Gandhi et al., "Learning behaviors from a single video demonstration using human feedback," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst.*, 2019, pp. 1970–1972.

[4] P. Goyal, S. Niekum, and R. J. Mooney, “Using natural language for reward shaping in reinforcement learning,” 2019, *arXiv:1903.02020*. [Online]. Available: http://arxiv.org/abs/1903.02020

[5] B. Prakash et al., “Guiding safe reinforcement learning policies using structured language constraints,” in *Proc. SafeAI Workshop 34th AAAI Conf. Artif. Intell.*, 2020, pp. 1–9.

[6] B. Prakash, N. Waytowich, A. Ganesan, T. Oates, and T. Mohsenin, “Guiding safe reinforcement learning policies using structured language constraints,” UMBC Student Collection, Baltimore, MD, USA, Tech. Rep., Mar. 2020.

[7] A. Jafari et al., “SensorNet: A scalable and low-power deep convolutional neural network for multimodal data classification,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 1, pp. 274–287, Jan. 2019.

[8] M. Hosseini et al., “On the complexity reduction of dense layers from O(N^2) to O(NlogN) with cyclic sparsely connected layers,” in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[9] T. Abtahi et al., “Accelerating convolutional neural network with FFT on embedded hardware,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 9, pp. 1737–1749, Sep. 2018.

[10] A. Shiri et al., “Energy-efficient hardware for language guided reinforcement learning,” in *Proc. Great Lakes Symp. VLSI*, Sep. 2020, pp. 131–136.

[11] M. Chevalier-Boisvert, L. Willems, and S. Pal. (2018). *Minimalistic Grid-World Environment for Openai Gym*. [Online]. Available: https://github.com/maximecb/gym-minigrid

[12] M. Chevalier-Boisvert. (2018). *Gym-Miniworld Environment for Openai Gym*. [Online]. Available: https://github.com/maximecb/gym-miniworld

Aidin Shiri is pursuing a PhD with the Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD, USA. His research interests mainly focus on energy-efficient digital application-specific integrated circuits and field-programmable gate array hardware accelerator design for deep neural networks and machine learning algorithm for low-power embedded devices.

Arnab Neelim Mazumder is pursuing a PhD with the Computer Science and Electrical Engineering Department, University of Maryland, Baltimore County, Baltimore, MD, USA. His research interests focus on deep neural networks, FPGA and ASIC designs, and low-power embedded systems. Mazumder has a BS degree from Bangladesh.

Bharat Prakash is pursuing a PhD with the Computer Science and Electrical Engineering Department, University of Maryland, Baltimore County, Baltimore, MD, USA. His research interests include deep reinforcement learning (RL), safe RL, and human in the loop RL.

Houman Homayoun is an Associate Professor with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA, USA. He conducts research in hardware security and trust, data-intensive computing, and heterogeneous computing.

Nicholas R. Waytowich is a Machine Learning Research Scientist with the Human Research and Engineering Directorate, U.S. Army Research Laboratory, Adelphi, MD, USA. His research interests include human-in-the-loop machine learning, human-autonomy integration, and deep reinforcement learning. Waytowich has a PhD in biomedical engineering from Old Dominion University, Norfolk, VA, USA. He is a Member of IEEE and the IEEE Systems, Man, and Cybernetics Society.

Tinoosh Mohsenin is an Associate Professor with the Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD, USA, where she is also the Director of the Energy Efficient High Performance Computing Lab. Her research is focused on designing energy efficient embedded processors for machine learning and signal processing, knowledge extraction techniques for autonomous systems, wearable smart health monitoring, and embedded big data computing.

■ Direct questions and comments about this article to Aidin Shiri, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250 USA; aidin.shiri@umbc.edu.