This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2021.3063363, IEEE Design and Test



# A Hardware Accelerator for Language Guided Reinforcement Learning

Aidin Shiri*, Arnab Neelim Mazumder*, Bharat Prakash*, Houman Homayoun†, Nicholas R. Waytowich‡, and Tinoosh Mohsenin*

* Department of Computer Science & Electrical Engineering University of Maryland, Baltimore County
†University of California, Davis ‡US Army Research Laboratory

***Abstract*—Reinforcement learning (RL) has shown great performance in solving sequential decision-making problems. This paper proposes a framework to train RL agents conditioned on constraints that are in the form of structured language to improve training efficiency. We implemented an energy-efficient hardware accelerator to receive both images and text inputs that allows RL agents understand human language and act in real-world environments. A scalable and parallel hardware with different number of processing elements is implemented on both FPGA and ASIC that provides a balance between power consumption and performance. The post-layout ASIC design consumes 9.2 mW while providing 361 fps throughput.**

***Keywords*—Reinforcement Learning, Structured Language Instructions, Energy Efficiency, FPGA, ASIC.**

## I. Introduction and Related Works

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] and [3]. It is important to train the agent in a way that easily abides the human instructions. One scalable way to do this is by using language instructions. Recent work in using language to guide reinforcement learning agents have gained 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 reinforcement learning agents [6]. While hardware implementation of deep neural networks has gained lot of attention in recent years [7], [8], [9], 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 [10]. In this paper, we propose a hardware-friendly architecture for Reinforcement Learning which can interpret language instructions to act in real-world scenarios. This paper makes the following contributions:

- Propose a hardware-friendly architecture for Reinforcement Learning which can interpret structured language instructions.
- Evaluate the model learning efficiency using two real-world environments with different levels of complexity.
- Propose a scalable, parallel and parameterized hardware in Verilog HDL that can receive image inputs from environment and language constraints for deploying on the embedded devices.
- Implementation of the proposed work with different configurations of processing elements on low power Artix-7 FPGA and providing a detailed analysis on 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.

## II. Proposed Method

### 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 $(S, A, P, R, \gamma)$, 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 paper we propose an architecture similar to the conventional Reinforcement Learning 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2021.3063363, IEEE Design and Test



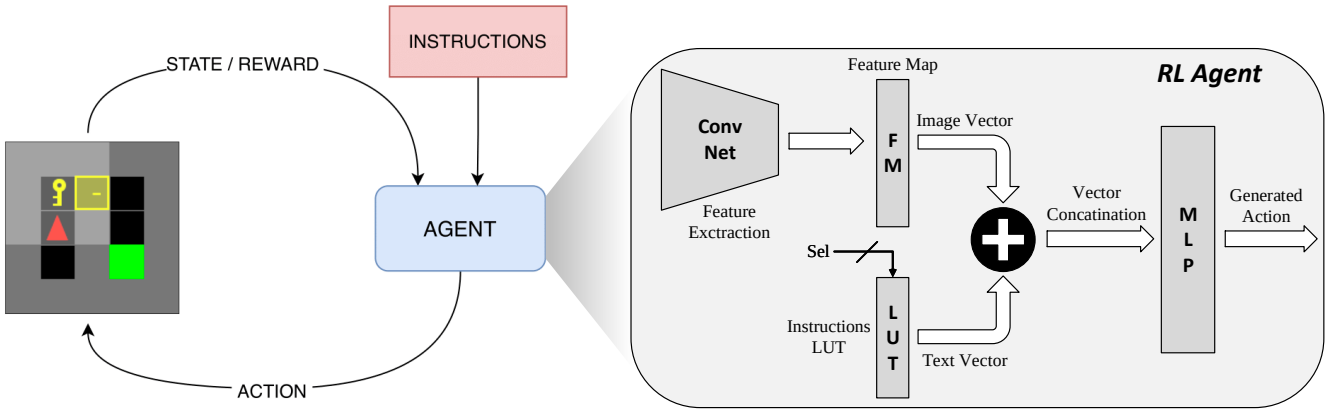



Fig. 1: The proposed architecture of the RL agent with language instructions for interacting with the environment while getting structured language constraints. In the conventional Reinforcement Learning model, agents interact with the environment through a set of actions and try to learn a certain policy 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.

### *B. Proposed System Architecture*

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. 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 reinforcement learning with the addition of an "Instructions" module during inference. The proposed system architecture consists of two main parts: the traditional Reinforcement Learning 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 to 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 convolution 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. 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 that can feed the saved instruction vectors to the agent.

## III. 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 small 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.

### *A. 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 4 directions, opening and closing doors and picking up and dropping objects. The environment is partially observable, the agent can only see an ego-centric $5 \times 5$ 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 baseline where we shape the environment rewards directly by giving negative rewards for violations.

The 2 Rooms scenario is shown in Fig. 2 (a)-up. It consists of 2 rooms separated by a wall and 2 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 I. 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).

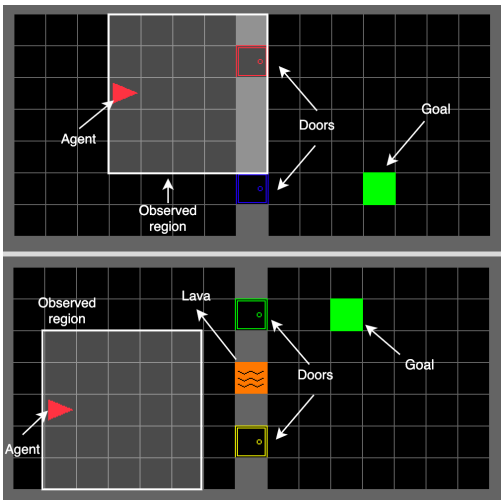
The 2 Rooms with lava scenario is shown in Fig 2. (a)-down. This is similar to the 2 Rooms environment but it has an additional cell called the Lava. It is the orange cell seen in the figure and it can behave in one of two ways depending on the language instruction. Example language instructions are shown in Table I. For the instruction of type 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 type 4-5, the agent entering the lava will die and result in 0 reward.

### *B. Case Study two: Miniworld*

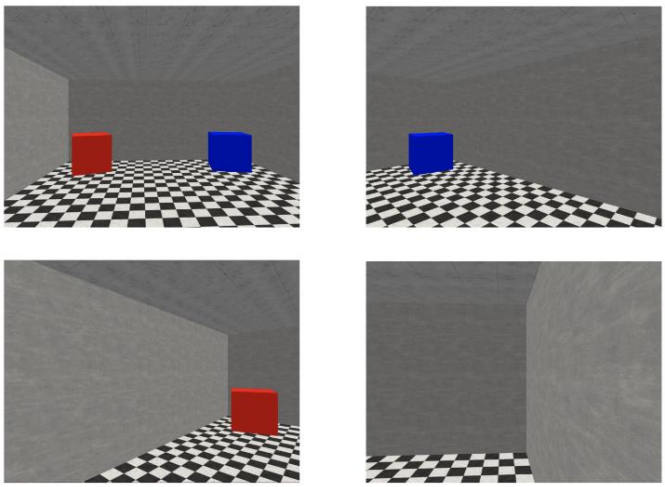
To observe the performance of the model in a more complex environment, we evaluated its performance in the MiniWorld

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2021.3063363, IEEE Design and Test

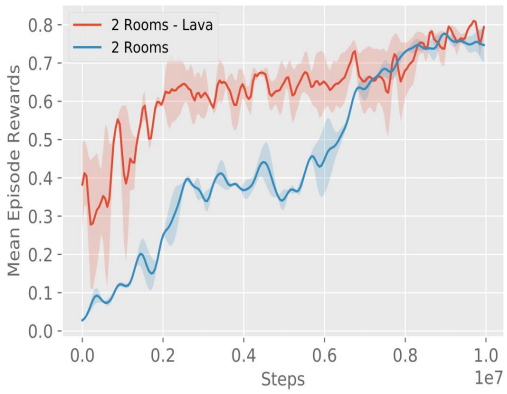




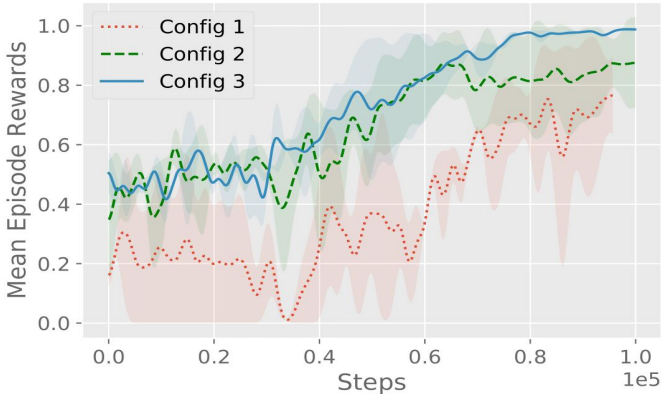

**(a) MiniGrid Environment, 2 Rooms and 2 Rooms-Lava**



**(b) Miniworld Environment, four different views from Agent Perspective**




**(c) MiniGrid Mean Episode Rewards**




**(d) Miniworld Mean Episode Rewards for three different network configurations**

Fig. 2: Two real-world environments with different complexities that are used to evaluate the training performance of our model.

environment as well. Second sets of experiments are performed on the modified MiniWorld environment. MiniWorld 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-(b). In our experiments, we set up a simple scenario where the agent is in a room and there are 2 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 shown in Table I. 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 2x2 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 paper is to propose efficient embedded hardware for reinforcement learning 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. In order 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2021.3063363, IEEE Design and Test



TABLE I: 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 |

The first, second, and third convolutional layers are tested with the configuration of 8, 16, and 32 filter sets respectively. Table II summarizes the number of parameters, computations, and required memory space for each configuration, and Fig. 2-(d) 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 2x less memory and 3x less computation. Therefore, we select Config 2 for implementation on the hardware.

## IV. Proposed Hardware Accelerator

### A. Accelerator Architecture Design

The block diagram of the proposed hardware architecture is illustrated in Fig. 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 processing elements (PE), 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) processing elements 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 processing elements with proper data. (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 is fed to the convolution block which calculates

TABLE II: 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 |

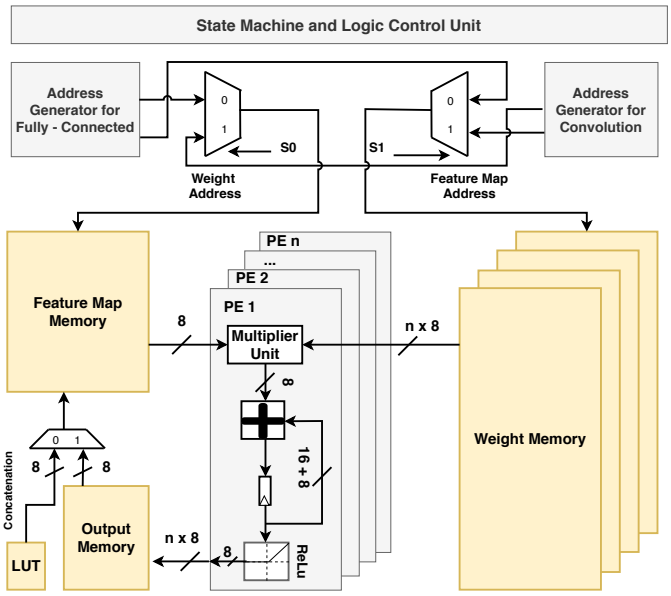



Fig. 3: Block diagram of hardware architecture which consists of feature map memory and weight memory that are accessed by the convolution and fully-connected address control unit to fetch data for the Processing Elements (PEs).

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 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 lookup table (LUT) is concatenated with the output of the first fully connected layer, and form the input of the next fully connected layer. As discussed in previous sections, a lookup table memory with a pre-trained 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 micro-architectures 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.

TABLE III: 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** |

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2021.3063363, IEEE Design and Test



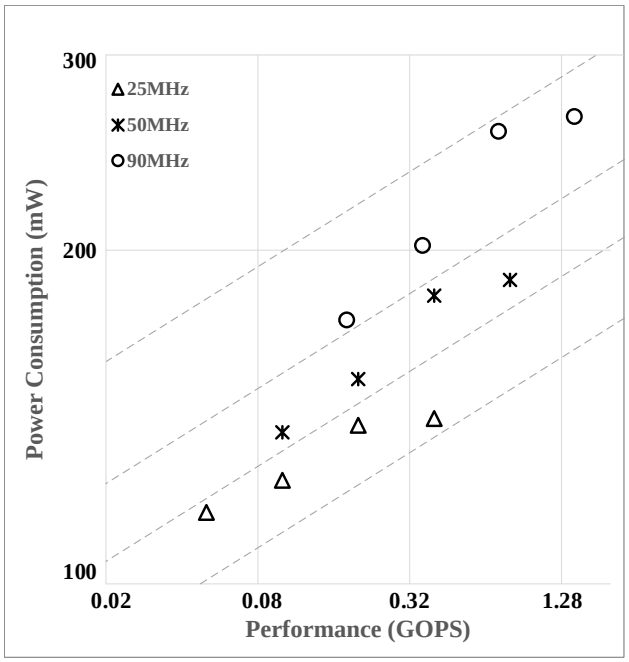



Fig. 4: Scatter plot of energy efficiency trade-off between performance and power for the FPGA implementation results. Dynamic power and performance are sketched with respect to the different frequencies and number of PE. Most energy efficient architecture-8 PE-is selected for ASIC implementation.

### *B. 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 8 processing elements 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.

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 III, it can be observed that while lower frequencies and number of PEs, yield lower power consumption, increasing them result in better energy efficiency. Figure 4 shows the scatter plot of hardware energy efficiency trade-off 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 8 number of PE, as an Application Specific Integrated Circuit (ASIC). A standard cell register transfer level (RTL) to 14nm FinFET post-layout with 0.8V 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 29x reduction in power consumption and 48x improvement of energy-efficiency compared to the FPGA implementation results. The details of ASIC implementation are provided in Table IV and the layout is shown in Fig. 5.

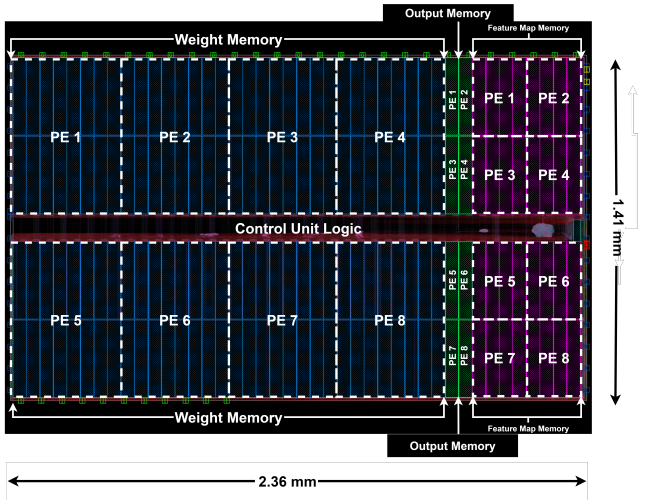



Fig. 5: Post-layout view of the proposed hardware accelerator for ASIC implementation in 14 nm, FinFET technology with the operating frequency of 150 MHz.

TABLE IV: Results of post place and route ASIC implementation of the proposed hardware accelerator with 8-bit memory width and 8 PEs in 14nm 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 |

## V. Conclusion

In this work, we proposed an energy-efficient hardware architecture for reinforcement learning 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 config-

ured to achieve high throughput or low power consumption. Multiple architectures with different parameters implemented on FPGA to find the most energy-efficient configuration and the best case is selected for ASIC implementation. While we emphasize on the energy-efficiency in deployment of reinforcement learning, 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 reinforcement learning environments and frameworks will result in great demand for energy efficient embedded reinforcement learning applications and facilitate future research in this direction.

## VI. Acknowledgment

This project was sponsored by the U.S. Army Research Laboratory under Cooperative Agreement Number 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, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2, no. 4.

[2] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, 2017, pp. 4299–4307.

[3] S. Gandhi, T. Oates, T. Mohsenin, and N. R. Waytowich, "Learning behaviors from a single video demonstration using human feedback," 2019.

[4] P. Goyal, S. Niekum, and R. J. Mooney, "Using natural language for reward shaping in reinforcement learning," *arXiv preprint arXiv:1903.02020*, 2019.

[5] B. Prakash *et al.*, "Guiding safe reinforcement learning policies using structured language constraints," in *SafeAI workshop Thirty-Fourth AAAI Conference on Artificial Intelligence*. AAAI, 2020.

[6] B. Prakash, N. Waytowich, A. Ganesan, T. Oates, and T. Mohsenin, "Guiding safe reinforcement learning policies using structured language constraints."

[7] A. Jafari, A. Ganesan, C. S. K. Thalisetty, V. Sivasubramanian, T. Oates, and T. Mohsenin, "Sensornet: A scalable and low-power deep convolutional neural network for multimodal data classification," *IEEE Transactions on Circuits and Systems I: Regular Papers*, no. 99, pp. 1–14, 2018.

[8] M. Hosseini, M. Horton *et al.*, "On the complexity reduction of dense layers from $o(n^2)$ to $o(nlogn)$ with cyclic sparsely connected layers," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019.

[9] T. Abtahi *et al.*, "Accelerating convolutional neural network with fft on embedded hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1737–1749, Sept. 2018.

[10] A. Shiri, A. N. Mazumder, B. Prakash, N. K. Manjunath, H. Homayoun, A. Sasan, N. R. Waytowich, and T. Mohsenin, "Energy-efficient hardware for language guided reinforcement learning," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 131–136.

[11] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for openai gym," https://github.com/maximecb/gym-minigrid, 2018.

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

**Aidin Shiri** is currently pursuing the Ph.D. degree with the Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 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 his Ph.D. degree in the Computer Science and Electrical Engineering Department at the University of Maryland Baltimore County, MD, USA. He received his B.S degree from Bangladesh. His research interests focus on deep neural networks, FPGA and ASIC designs, and low-power embedded systems.

**Bharat Prakash** is PhD student in the Computer Science and Electrical Engineering Department at the University of Maryland Baltimore County, MD, USA. His research interests include Deep Reinforcement Learning (RL), Safe RL and Human in the loop RL.

**Houman Homayoun** is currently an Associate Professor in the Department of Electrical and Computer Engineering at University of California, Davis. He conducts research in hardware security and trust, data-intensive computing and heterogeneous computing, where he has published more than 150 technical papers in the prestigious conferences and journals on the subject and directed over $9M in research funding from NSF, DARPA, AFRL, NIST and various industrial sponsors.

**Nicholas Waytowich** is a machine learning research scientist at the Human Research and Engineering Directorate at the US Army Research Laboratory. His research interests include human-in-the-loop machine learning, human-autonomy integration, and deep reinforcement learning. Waytowich received a PhD in biomedical engineering from Old Dominion University. He is a member of IEEE and the IEEE Systems, Man, and Cybernetics Society.

**Tinoosh Mohsenin** is currently an Associate Professor with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, where she is also the Director of the Energy Efficient High Performance Computing Lab. She has authored or co-authored over 130 peer-reviewed journal and conference publications. Her research focus is 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.

**Contact Information** Direct questions and comments about this article to Aidin Shiri, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD, 21250; e-mail: aidin.shiri@umbc.edu