ARL-TR-10210 ● SEP 2025

DEVCOM ARMY RESEARCH LABORATORY

HarDRL: Acceleration of Deep Reinforcement Learning Algorithms in Hardware

by Vinod K. Mishra and Kanad Basu

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-10210 ● SEP 2025

HarDRL: Acceleration of Deep Reinforcement Learning Algorithms in Hardware

Vinod K. Mishra
DEVCOM Army Research Laboratory

Kanad Basu
The University of Texas at Dallas

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
September 2025	Technical Report	7/1/2024	12/31/2024

4. TITLE AND SUBTITLE
HarDRL: Acceleration of Deep Reinforcement Learning Algorithms in Hardware

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Vinod K. Mishra and Kanad Basu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
DEVCOM Army Research Laboratory ATTN: FCDD-RLA-NC Aberdeen Proving Ground, MD 21005	ARL-TR-10210

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The proliferation of Deep Reinforcement Learning (DRL) has led to its adoption in solving several real-world problems (e.g., medical diagnosis, drug discovery, and self-driving vehicles). Furthermore, in recent years, DRL algorithms have been incorporated into military and defense applications such as remote surgical and evacuation support, cybersecurity, and target recognition. However, because conventional DRL implementations use general-purpose CPUs and graphical processing units, they are resource intensive. They incur significant power, area, and latency overheads, leading to inefficiency and inefficacy. As a solution to these problems, we propose hardware (HW) acceleration of DRL algorithms for real-time deployment. This involves the 8-bit quantization of model parameters and inputs, followed by novel dictionary-based compression and HW-based decompression strategies, resulting in more than four-times reduction in memory footprint. This leads to a maximum memory savings of 80.73% while incurring a negligible area overhead of 0.022% and power overhead of 0.002%, thereby facilitating the efficient deployment of DRL algorithms in HW.

15. SUBJECT TERMS
Deep Reinforcement Learning; hardware acceleration; dictionary-based compression; hardware-based decompression; Network, Cyber, and Computational Sciences

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	21

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Vinod K. Mishra	(520) 691-5761

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Introduction

In recent years, Deep Reinforcement Learning (DRL) algorithms have gained significant traction in a wide range of applications, particularly in real-time control and policy-based decision-making for autonomous vehicles, robots, and drones.[1,2] Although these algorithms have demonstrated great potential in those domains, it remains important to explore their potential for defense scenarios, especially in army-specific situations such as tactical arenas. DRL implementations typically rely on general purpose CPUs and graphical processing units (GPUs), with optimization efforts focused on software; for instance, optimizing algorithms by determining the best policy and architecture for a given application.[1]

Recently, however, researchers have emphasized the optimization of hardware (HW) implementation of ML workloads related to both deep neural network (DNN) and DRL models.[3–7] In the case of DNNs, existing research has primarily focused on minimizing the energy consumption during computation.[3,4] However, since DNNs are characterized by a large memory footprint, the energy overheads associated with their memory subsystem are considerably higher than the computation subsystem.[8,9] This can render inefficiency during the inference execution of DNNs in resource-constrained internet-of-things (IoT) edge devices, and, to this end, researchers have proposed low-power, HW-based compression strategies[9,10] (as shown in Figure 1).

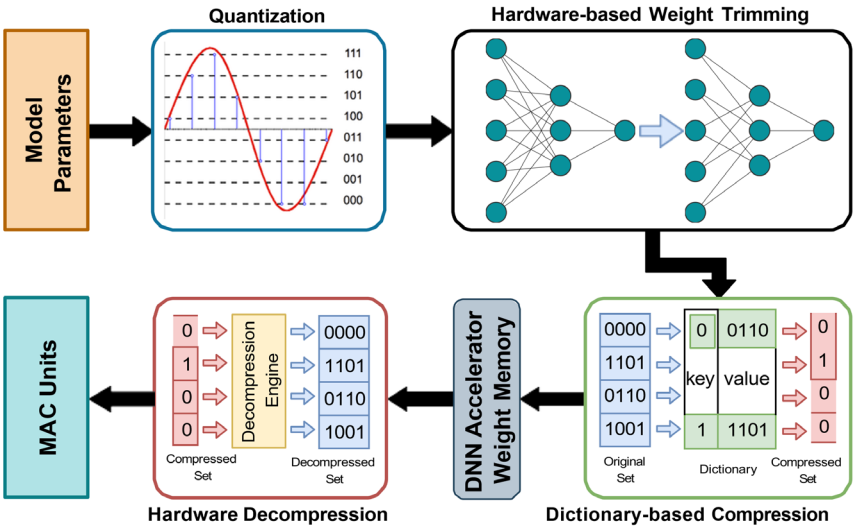

Figure 1. Overview of the proposed approach.

Like DNNs, DRL models are associated with a large memory footprint as well. Although recent research has focused on optimizing the HW implementation of DRL algorithms,[5–7] the proposed solutions have the following limitations:

- Restricted to smaller models and specific applications,
- Resource-intensive, and
- Dependent on large-bit representation of model parameters, such as 32-bit floating point and 16-bit fixed point.

Although HW accelerators[7] have been explored for HW implementation of DRL algorithms, there is a dearth of a generic solution that facilitates efficient DRL deployment in resource-constrained IoT edge devices. To this end, we propose a novel strategy to accelerate the deployment of DRL workloads in HW, as illustrated in Figure 1. Specifically, this includes the following:

- Quantization of model parameters and inputs furnishing up to four-times reduction in memory requirement,
- Dictionary-based compression strategy furnishing additional memory savings, and
- A dedicated HW module enabling low-power decompression.

In addition, this approach

- Presents the first HW-based compression strategy (to the best of our knowledge) for accelerated deployment of DRL algorithms,
- Includes quantization producing four-times reduction in memory requirement, HW-based trimming of model weights, dictionary-based compression, and low-power decompression in HW, and
- Evaluates the new system in three DRL environments, achieving a reduction in memory footprint of 80.73% while incurring a negligible area overhead of 0.022% and power overhead of 0.002%.

We provide background and motivation in Section 2, describe the proposed approach in Section 3, discuss the experimental setup and results in Section 4, and finally summarize in Section 5.

2. Background and Motivation

DRL techniques are increasingly used to address various real-world challenges, ranging from object detection and face recognition to natural language processing and data mining. One of the major advantages of DRL models over traditional deep learning models is their capacity to learn and train in dynamic real-world settings while autonomously extracting features. However, like deep learning models, the

conventional CPU, GPU, and cloud-based implementations of DRL models result in significant overhead regarding computation, power, and latency. For instance, DeepMind's AlphaGo, a reinforcement learning (RL) program developed for playing the game of Go, requires 1920 CPUs, 280 GPUs, and several weeks to train, consuming 1 MW of power, equivalent to that of 50,000 human brains.[11] Such extensive resources are not practical for real-time deployment at the edge.

Therefore, it is imperative to reduce the power consumption and costs of CPUs and GPUs associated with implementing DRL applications. Customized HW accelerators such as Google Tensor Processing Unit and Intel's Neural Processing Unit have been developed to tackle these issues by speeding up the computationally intensive matrix multiplication operations in DNNs.[12] However, there is a lack of an accelerator that addresses these challenges specifically in DRL models. In addition to the previously mentioned challenges, the implementation of DRL models in real-world scenarios is associated with obstacles, such as dealing with high-dimensional continuous state-action spaces (i.e., large hyperparameter space), intricate training procedures, limited data samples, real-time inference, high latency, and partial observability.[10]

3. Proposed Methodology

In this section, we provide details of our approach. The flow of operations with explanations for proposed sequence emphasizes the technical steps needed.

3.1 Operation Flow and Proposed Sequence

We provide an overview of the proposed strategy pertaining to the acceleration of DRL models in HW. The sequence of operations involved in this approach is demonstrated in Algorithm 1.

Algorithm 1 DRL Hardware Acceleration: Sequence of Operations

Input: $absWeights$, $inputAct$, b
Output: $unWeights$

1: $quantAct \leftarrow Quantize(inputAct, b)$
2: $quantWeights \leftarrow Quantize(absWeights, b)$
3: $trimWeights \leftarrow WeightTrim(quantWeights)$
4: $dictVals \leftarrow DictionaryCompress(trimWeights)$
5: $unWeights \leftarrow HardwareDecompress(dictVals)$

The inputs to this algorithm are model weights (*absWeights*), input activations (*inputAct*), and quantization bit length (b), whereas the output is the set of uncompressed weights (*unWeights*). The first process involves quantization of the input activations and model weights from the existing 32-bit floating point representation to the desired b-bit integer representation, as illustrated in lines 1 and 2. The next step addresses the HW-based trimming of the quantized weights, as shown in line 3. After this step, we perform dictionary-based compression to generate dictionary entries, which are a subset of quantized weights that are representative of the entire set of model weights (line 4). These compressed weights are subsequently stored in the weight memory of the DNN accelerator. The final step involves the HW-based decompression of these compressed weights into their uncompressed form, as shown in line 5. These weights (*unWeights*) are subsequently mapped to the multiply and accumulate (MAC) units before initiating the inference phase. The detailed description of the various steps is provided below.

3.2 HW-Based Weight Trimming

The first stage focuses on the HW-based weight trimming, as illustrated in Figure 2.

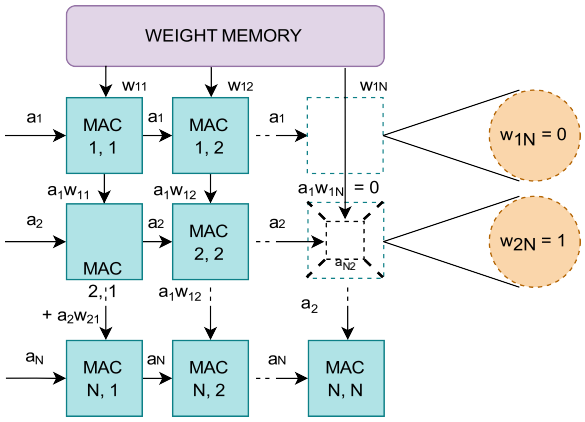

Figure 2. Overview of the HW-based trimming approach.

The quantized weight values are assumed to be stored in the accelerator memory, similar to previous works.[13–15] If a weight value of 0 is encountered, the corresponding MAC unit is bypassed because passing a value of 0 to a MAC unit is equivalent to disabling it. As shown in Figure 2, $MAC_{1,N}$ is bypassed because it is mapped to weight $w_{1N} = 0$. If a weight value of 1 is encountered, we bypass the multiplier of the corresponding MAC unit, making it function only as an adder. This is because the result of the multiplier is identical to the input activation when the weight is 1. Therefore, we add the input activation directly to the output of the previous MAC unit in the same column. As illustrated in Figure 2, $MAC_{2,N}$ is

mapped to weight $w_{2N} = 1$, so we bypass its multiplier and add its input a_2 to the output of $MAC_{1,N}$, resulting in a_2. This means that weights of values 0 and 1 do not need to be stored in the weight memory. Since these weights are known to constitute a significant portion of large DNN networks, this step results in a substantial reduction in memory traffic.[10]

3.3 Dictionary-Based Compression

The second step involves dictionary-based compression. Specifically, there are two types of compression: lossy and lossless compression.[10] Lossless compression does not compromise the accuracy of the data, but it may not always provide a significant level of compression. Conversely, although lossy compression offers a predictable level of compression, it is associated with performance degradation or accuracy loss. For lossless compression, the most frequently occurring weight values are selected as the dictionary entries, like that shown by Seong et al.[16] However, while populating the dictionary for lossy compression, we need to consider the correlation between mismatched weights and dictionary entries, along with the frequency of occurrence of the weights. An illustrative example of both compression techniques is depicted in Figure 3.

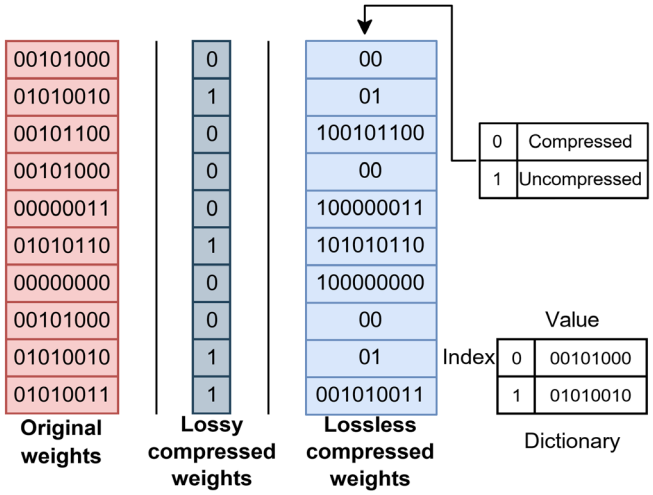

Figure 3. Illustrative example of the dictionary-based compression technique.

Here, we consider ten 8-bit weights (totaling 80 bits) and a dictionary having two entries that can be represented using only one bit. For lossy compression, each mismatched weight is replaced by the appropriate dictionary index (index of dictionary entry closest in value to the mismatched weight). On the other hand, weights having a match in the dictionary are represented by the corresponding dictionary index. For instance, the second weight (01010010) matches the second dictionary entry and is thus represented using its index (1). On the contrary, the

seventh weight (00000000) is closest in value to the first dictionary entry and is represented as 0. As evident from Figure 3, lossy compression reduces the size of the weight set to 10 bits, thereby furnishing significant memory savings of 87.5%. In the case of lossless compression, if a weight has a match with one of the dictionary entries, we first set its most significant bit (MSB) to 0, following which we set its least significant bit to the appropriate dictionary index, thereby resulting in a 2-bit value. For mismatched weights, their original 8-bit value MSB is prefixed by 1, thereby increasing the number of bits to 9. In comparison to the lossy compression approach, lossless compression furnishes a value of 01 for the second weight and 100000000 for the seventh entry. Lossless compression yields a 55-bit weight set, thereby reducing the memory requirement by 31.25%. Despite inducing minor performance degradation, we consider lossy compression due to the higher compression savings associated with it.[9,10] For this scenario, we consider both the frequency of weights and approximation using Euclidean distance for dictionary selection, like that shown by Arunachalam et al.[10]

Algorithm 2 demonstrates the proposed lossy compression-based dictionary selection strategy, which corresponds to the *DictionaryCompress* function described in Algorithm 1.

Algorithm 2 Compressed Dictionary Selection

Input: *quantWeights*, *b*, *L*
Output: *dictVals*

```
1:    for L iterations do
2:    for i in range(0, 2^b − 1) do
3:    for j in quantWeights do
4:    selWt[i] += abs(i - j)
5:    end for
6:    end for
7:    if dictVals is empty then
8:    minValue ← min(selWt)
9:    else if dictVals is not empty then
10:   selWt[i] += max(selWt) for i in dictVals
11:   goto 8
12:   end if
13:   end for
```

While the inputs are quantized model weights (*quantWeights*) (obtained as output from line 1 of Algorithm 1), quantization bit length (*b*), and desired dictionary length (*L*), the output is a set of dictionary entries (*dictVals*) (output specified in line 4 in Algorithm 1). Our goal is to identify the best subset of *L* weights are representative of the entire set of model weights. To do so, we calculate the sum of the Euclidean distance of all weights from each index value (stored in a variable

called *selWt*), as shown in lines 2 through 6. The indices range from 0 to $2^b - 1$; since we are dealing with 8-bit ($b = 8$) quantized weights in this work, this is equivalent to a range between 0 and 255. The first and last entries of *selWt*, that is, *selWt*[0] and *selWt*[255], represent the proximity of weight set *quantWeights* to index 0 and index 255, respectively. As mentioned earlier, our algorithm is based on minimizing the Euclidean distance between the dictionary entries and mismatched weights. Therefore, we subsequently select the index having the minimum Euclidean distance as the dictionary entry, as shown in line 8. To avoid repeated dictionary entries, we include a fail-safe condition by adding a penalty of max(*selWt*) to the Euclidean sum of existing dictionary entries, following which we determine the new dictionary entry, as demonstrated in lines 10 and 11. We continue this process for L iterations, thereby obtaining the desired L dictionary entries.

3.4 Hardware Decompression

Before being used by the DNN accelerator for inference, the compressed weights (stored in the accelerator's weight memory) need to be decompressed to their original quantized values. This operation requires N decompression HW units (DHUs) for an $N \times N$ systolic array-based accelerator, with each DHU assigned to a column. Each DHU is responsible for decompressing weights in real-time and forwarding them to the corresponding MAC unit in the same column. The net decompression area overhead in an $N \times N$ systolic array-based DNN accelerator is calculated as d/Nm, where d and m are area overheads of a DHU and MAC unit, respectively.

4. Experimental Results

4.1 Experimental Setup

The proposed approach is evaluated on three case studies involving classic RL environments of ***cartpole***, ***acrobot***, and ***mountain car*** using a Deep Q Network (DQN). The details of each case study are described in the following sections.

Cartpole**:** In the cartpole system, a pole is affixed to a cart, forming an inverted pendulum with an unstable center of gravity above its pivot point.[2] To maintain balance, an appropriate force must be applied to either the cart or the pivot point. The equations of motion can be obtained from the system's Hamiltonian dynamics and then solved numerically. The **cartpole** system is an experimental setup for evaluating the performance of DRL algorithms. Here, the objective is to keep the pole upright by moving the cart left or right. The parameters associated with this

case study are as follows: 1) the cart's position, 2) its velocity, 3) the pole's angular position, and 4) its angular velocity.

***Acrobot*:** The second scenario is the **acrobot** system consisting of a two-link robotic arm located in a vertical plane with an actuator located at the intersection of the two arms.[17] The top arm lacks an actuator at the shoulder, which distinguishes it from the **pendubot** model, where the actuator is situated at the shoulder but not at the elbow. The most common control task for the **acrobot** is to swing-up, where it leverages the torque at the elbow to move into a vertical configuration and subsequently maintain balance. The objective here is to reach a predetermined height by adjusting the movement of the bottom link of the robot, either by moving it to the left, right, or keeping it still. Moreover, the parameters considered are the position and velocity of the robot, and the evaluation metric is the policy's average return for all training iterations. The policy is enhanced by implementing the following measures:

- Increase the average return of the policy with each training iteration,
- Impose a penalty of 1 for each move,
- Limit the maximum number of movements per iteration to 500, and
- Set the maximum penalty (minimum reward) to −500.

***Mountain Car*:** The third case study of a **mountain car** is set in an environment requiring the car to escape from a valley.[18] The car must escape a valley by reversing up and gaining sufficient momentum to climb up a mountain (owing to its underpowered engine). The parameters considered in this case study are the car's position and its velocity. There are three actions associated with the **mountain car** problem: forward acceleration, backward acceleration, and zero throttle. The primary objective is to minimize the number of steps taken by the car, whereas the secondary objectives include minimizing the number of backward and forward acceleration actions. In this regard, each time step and each backward or forward acceleration action are penalized with −1. Hence, this case study is associated with three intrinsic rewards, one for each objective. The implementation limits the number of movements per iteration to 200 and the maximum penalty (reward) to −200.

4.2 Results

We give the results obtained for the three systems with details obtained through our technical approach. The results for the **cartpole** environment are demonstrated in Table 1.

Table 1. Compression savings for the cartpole experiment.

Dictionary length	Compression savings: Q + C (%)	Compression savings: Q + HT + C (%)
1	−9.45	−3.56
2	−6.81	−1.07
4	−3.07	2.58
8	2.75	8.17
16	10.94	16.16
32	17.55	21.95
64	18.41	22.92

Note: C = compression; HT = HW-based weight trimming; Q = quantization.

The first column indicates the dictionary length (L); column two outlines the memory savings furnished due to quantization and dictionary-based compression (Q + C); and column three represents the memory savings due to quantization, HW-based weight trimming and dictionary-based compression (Q + HT + C). Since our initial results indicated larger savings (which is also demonstrated by existing research),[9,10] we consider lossy compression over its lossless counterpart. Moreover, the accuracy loss associated with this type of compression is negligible (0.5%).

As evident from Table 1, the compression savings increase with dictionary length in both columns two and three. Although Q + C furnishes a maximum memory savings of 18.41% for L = 64, Q + HT + C furnishes higher savings of 22.92% at L = 64. The reported compression savings are in addition to the four-times memory reduction furnished by the quantization of weights and activations from 32-bit floating point to 8-bit integer representation. Therefore, the proposed approach produces a cumulative memory savings of 80.73% for the cartpole experiment. The experimental results for **acrobot** are shown in Table 2.

Table 2. Compression savings for the acrobot system.

Dictionary length	Compression savings: Q + C (%)	Compression savings: Q + HT + C (%)
1	−7.3	−3.7
2	−5.1	−1.7
4	−2.15	1.1
8	2.2	5.5
16	7.5	10.48
32	13.3	16.3
64	16.38	19.15

While column one represents the dictionary length (L), the second and third columns demonstrate the memory savings due to Q + C and Q + HT + C, respectively. The performance degradation associated with lossy compression is negligible (0.65%) for the **acrobot** environment. From Table 2, it is evident that

compression savings furnished by Q + HT + C is higher than Q + C. Furthermore, it increases with dictionary length for both scenarios. Although Q + C furnishes a maximum compression savings of 16.38% at L = 64, it is outperformed by Q + HT + C, which produces higher savings of 19.35% for the same dictionary length of L = 64. As mentioned in the previous section, the compression savings values reported here are in addition to the four-times memory reduction furnished by the 8-bit quantization of weights and activations. Hence, the total memory savings furnished by the proposed approach is 79.84% for the cartpole experiment. The results for the **mountain car** environment are shown in Table 3.

Table 3. Compression savings for the mountain car experiment.

Dictionary length	Compression savings: Q + C (%)	Compression savings: Q + HT + C (%)
1	−11.1	−6.59
2	−9.9	−4.33
4	−8.13	−1.9
8	−5.6	2.52
16	−1.75	7.8
32	2.5	13.26
64	6.1	16.2

Like the previous two tables, column one is associated with the dictionary length (L), whereas columns two and three correspond to the compression savings due to Q + C and Q +HT +C, respectively. In this scenario, the accuracy drop due to lossy compression is marginal (1.1%). As illustrated in Table 3, we obtain significantly higher memory savings due to Q + HT + C than Q + C, wherein both cases exhibit a positive trend of savings versus dictionary length, that is, *increase in compression savings with incremental dictionary lengths*. Although Q + C produces a maximum memory savings of 6.1% at L = 64, its Q + HT + C counterpart furnishes significantly higher savings of 16.2%, both of which are add-ons to the four-times memory savings furnished by the 8-bit quantization of model weights and inputs. Thus, the total memory savings furnished in the mountain car scenario is 79.05%.

These experiments show that the proposed dictionary-based compression strategy furnishes considerable memory savings in lieu of minimal performance degradation. Thus, our approach can be incorporated to facilitate efficient and reliable DRL inference in HW. Next, we evaluate the HW overheads associated with the proposed decompression strategy.

4.3 HW Decompression Overhead

In this section, we outline the overheads incurred to implement decompression in HW (DNN accelerator). First, the decompression HW is designed and synthesized using Verilog and Synopsys Design Compiler (using lsi_10k and sead_90nm digital standard cell libraries), respectively. The area and power overheads are illustrated in Figures 4a and b, respectively.

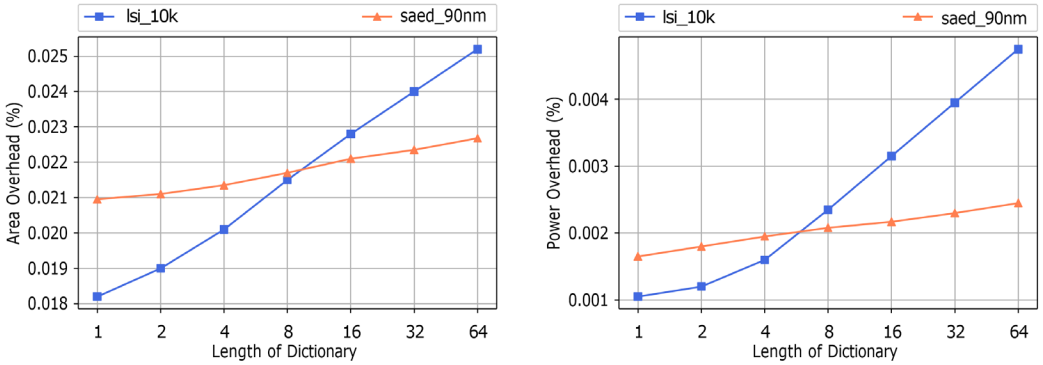

Figure 4. Variation of decompression (a) area and (b) power overheads with incremental dictionary lengths.

As evident from Figure 4, both overheads demonstrate a linear relationship with the dictionary length (L). While the maximum dictionary length (L = 64) incurs an area overhead of 0.022% and power overhead of 0.002% for **saed_90nm** library, it furnishes area and power overheads of 0.025% and 0.004% for **lsi_10k**. Therefore, the proposed HW-based decompression approach incurs negligible HW overheads, thereby facilitating an efficient and resource-friendly DRL inference operation.

5. Conclusion

In this report, we propose a novel strategy for optimizing the HW deployment of DRL models. Our proposed approach involves 8-bit quantization of model weights and inputs followed by HW-based trimming of model weights, dictionary-based compression, and HW decompression. We evaluated our strategy using a DQN on classic DRL tasks, such as cartpole, acrobat, and mountain car environments. The proposed approach furnishes a maximum memory savings of 80.73% while incurring negligible area overhead of 0.022% and power overhead of 0.002%, thereby facilitating an efficient deployment of DRL algorithms in HW.

6. References

1. Dulac-Arnold G, Mankowitz D, Hester T. Challenges of real-world reinforcement learning. arXiv; 2019. arXiv:1904.12901. https://doi.org/10.48550/arXiv.1904.12901

2. Lillicrap TP et al. Continuous control with deep reinforcement learning. arXiv; 2015. arXiv:1509.02971. https://doi.org/10.48550/arXiv.1509.02971

3. Mazahir S, Hasan O, Shafique M. Adaptive approximate computing in arithmetic datapaths. IEEE Design & Test. 2017;35(4):65–74.

4. Shafique M, Hafiz R, Rehman S, El-Harouni W, Henkel J. Cross-layer approximate computing: from logic to architectures. DAC 2016: Proceedings of the 53rd Annual Design Automation Conference; 2016 June 5–9; Austin, TX. IEEE; c2016; p. 1–6. https://doi.org/10.1145/2897937.2906199

5. Shiri A et al. An energy-efficient hardware accelerator for hierarchical deep reinforcement learning. 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS); 2021 June 6–9; Washington, DC. IEEE; c2021. p. 1–4.

6. Fu Y, Zhang Y, Li C, Yu Z, Lin YC. A3C-S: automated agent accelerator co-search towards efficient deep reinforcement learning. 2021 58th ACM/IEEE Design Automation Conference (DAC); 2021 Dec 5–9; San Francisco, CA. IEEE; c2021 p. 13–18.

7. Yang J, Hong S, Kim J-Y. FIXAR: a fixed-point deep reinforcement learning platform with quantization-aware training and adaptive parallelism. 2021 58th ACM/IEEE Design Automation Conference (DAC); 2021 Dec 5–9; San Francisco, CA. IEEE; c2021. p. 259–264.

8. Ghosh SK, Raha A, Raghunathan V. Approximate inference systems (AxIS) end-to-end approximations for energy-efficient inference at the edge. Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design; 2020 Aug 10–12; Virtual. c2020. p. 7–12. https://doi.org/10.1145/3370748.3406575

9. Arunachalam A et al. A novel low-power compression scheme for systolic array-based deep learning accelerators. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2023;42(4)1085–1098.

10. Arunachalam A et al. HardCompress: a novel hardware-based low-power compression scheme for DNN accelerators. 2021 22nd International

Symposium on Quality Electronic Design (ISQED); 2021 Apr 7–9; Santa Clara, CA. IEEE; c2021. p. 457–462.

11. Silver D et al. Mastering the game of go with deep neural networks and tree search. Nature. 2016;529(7587):484–489.

12. Jouppi NP et al. In-datacenter performance analysis of a tensor processing unit. Proceedings of the 44th Annual International Symposium on Computer Architecture; 2017 June 24–28; Toronto, ON, Canada. IEEE; c2017; p. 1–12.

13. Zhang JJ et al. Fault-tolerant systolic array based accelerators for deep neural network execution. IEEE Design & Test. 2019;36(5):44–53.

14. Wang K, Liu Z, Lin Y, Lin J, Han S. HAQ: hardware-aware automated quantization with mixed precision. Proceedings of the IEEE CVPR Conference on Computer Vision and Pattern Recognition (CVPR); 2019 June 15–20; Long Beach, CA. IEEE; c2019. p. 8612–8620.

15. Jacob B et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proceedings of the IEEE CVPR Conference on Computer Vision and Pattern Recognition (CVPR); 2018 June 18–23; Salt Lake City, UT; 2018. p. 2704–2713.

16. Seong S-W, Mishra P. Bitmask-based code compression for embedded systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2008;27(4):673–685.

17. Gillen S et al. Combining deep reinforcement learning and local control for the acrobot swing-up and balance task. 2020 59th IEEE Conference on Decision and Control (CDC); 2020 Dec 14–18; Jeju, South Korea. IEEE; c2020. p. 4129–4134.

18. Nguyen TT et al. A multi-objective deep reinforcement learning framework. Engineering Applications of Artificial Intelligence. 2020;96:103915.

List of Symbols, Abbreviations, and Acronyms

C	compression
CPU	central processing unit
DHU	decompression hardware unit
DNN	deep neural network
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
GPU	graphical processing unit
HT	hardware-based weight trimming
HW	hardware
IoT	internet-of-things
MAC	multiply and accumulate
ML	machine learning
MSB	most significant bit
Q	quantization
RL	reinforcement learning
TPU	Tensor Processing Unit

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CI TECH LIB