Action Evaluation Hardware Accelerator for Next-Generation Real-Time Reinforcement Learning in Emerging IoT Systems

Jianchi Sun[‡], Nikhilesh Sharma[†], Jacob Chakareski[*], Nicholas Mastronarde[†], Yingjie Lao[‡]
[‡]Clemson University, [†]University at Buffalo, [*]New Jersey Institute of Technology

***Abstract*—Internet of Things (IoT) sensors often operate in unknown dynamic environments comprising latency-sensitive data sources, dynamic processing loads, and communication channels of unknown statistics. Such settings represent a natural application domain of reinforcement learning (RL), which enables computing and learning decision policies online, with no a priori knowledge. In our previous work, we introduced a post-decision state (PDS) based RL framework, which considerably accelerates the rate of learning an optimal decision policy. The present paper formulates an efficient hardware architecture for the action evaluation step, which is the most computationally-intensive step in the PDS based learning framework. By leveraging the unique characteristics of PDS learning, we optimize its state value expectation and known cost computational blocks, to speed-up the overall computation. Our experiments show that the optimized circuit is 49 times faster than its software implementation counterpart, and six times faster than a Q-learning hardware accelerator.**

***Index Terms*—Reinforcement Learning, Hardware Acceleration, Wireless Communication, Action Evaluation, IoT Systems, Emerging Latency-Sensitive Applications.**

I. Introduction

A variety of emerging applications spanning autonomous driving, mobile augmented and virtual reality, remote multi-view sensing, personalized healthcare, virtual teleportation, UAV-IoT, 360° video streaming, remote robot navigation, cooperative video delivery, and telemetry [1–10], rely on computing and communication limited Internet of Things (IoT) devices and sensors [11–13]. The stochastic processes governing the captured latency-sensitive data and the channel dynamics, arising in such emerging settings, are not known a priori. This necessitates learning the respective desired optimal transmission policies online, during operation, to adapt to the experienced traffic and channel dynamics.

To this end, reinforcement learning (RL) [14, 15] has been shown to be an extremely effective tool, with Q-learning being its most widely-used method [16]. For instance, Q-learning has been employed to maximize the throughput of an energy-harvesting transmitter [17]. While Q-learning can solve problems with small state/action spaces, it exhibits poor convergence rates, which makes it inappropriate for problems involving large state/action spaces. Additionally, this approach is purely data-driven, which does not incorporate any useful information about the underlying system dynamics.

Recently, we explored and advanced the concept of *post-decision states* (PDS) [14, 15, 18–23], which exploits basic system knowledge to considerably advance the RL learning rate. PDS capture the system state after an action is taken, but before the unknown dynamics take place, which allows us to decompose the problem into *known* and *unknown* components, where only the latter must be learned. Though using PDS can speed-up the convergence to the optimal policy, it introduces the cost of increased action-selection complexity [15], which brings challenges to real-time applications. Moreover, the limited computing and power of wireless IoT systems [24] represent further challenges to actual deployment. Thus, hardware acceleration is a promising direction to enable real-time IoT applications of PDS based learning [25, 26].

In this paper, we design an efficient architecture for action evaluation, which computes the action to select given the present state. This step is the computational bottleneck in PDS based RL systems, as it is involved in greedy action selection and state value updating in each iteration. The key novelty of our design includes i) re-structuring the action evaluation of PDS based RL for hardware optimization, which yields a speed up of over 49 times, compared to the software counterpart; and ii) further optimizing the hardware accelerator's performance by efficiently computing the transmission power costs (P_{tx}) and packet loss rates (PLR) using lookup tables (LUTs), re-ordering the register array for the value function $V(s)$, and parallelizing the computation with two dedicated trees. As a result, the computational delay of our hardware accelerator is further reduced by 66.3%, while the power consumption and cells number are also decreased by 85% and 86%, respectively. Meanwhile, when compared to Q-learning, our optimized accelerator achieves a 83% delay reduction and a 59% power consumption reduction.

The rest of this paper is organized as follows. Section II reviews the mathematical background of PDS based RL and conventional Q-learning. Then, our proposed architecture is described in detail in Section III. Section IV presents the experimental results to verify the effectiveness of the proposed architectures. Finally, Section V concludes the paper.

The work of N. Sharma and N. Mastronarde was supported in part by the National Science Foundation (NSF) under Award ECCS-1711335. The work of J. Chakareski was supported in part by the NSF under Awards CCF-1528030, ECCS-1711592, CNS-1836909, and CNS-1821875.

II. Background

A. PDS based Reinforcement Learning

We consider a time-slotted wireless IoT sensor and aim at improving the wireless power management, with the specific objective to minimize the sensor's energy consumption, subject to an operational delay constraint.

To implement RL for the wireless power management problem, we first formulate it into a constrained MDP. We assume that time is divided into slots with length ΔT (seconds) and that the system's state in the n-th time slot is denoted by $s^n \triangleq (b^n, h^n, x^n) \in \mathcal{S}$, with packet buffer state b^n (i.e., the number of packets stored in the buffer), channel fading state h^n, and power management state x (radio on/off). At the beginning of each time slot, the IoT sensor observes its state s^n and takes an action $a^n = (BEP^n, y^n, z^n)$, where BEP^n is its target bit-error probability, y^n is its power management action (turn on/off the radio), and z^n is its packet throughput (number of transmitted packets). We aim to determine the action in each state to minimize the cost $c(s^n, a^n) = \rho(s^n, a^n) + \lambda g(s^n, a^n)$ over time, where $\rho(s, a)$ is the power cost, $g(s, a)$ is the delay cost, and λ is a Lagrange multiplier to set the delay constraint.

The sequence of states $s^n : \{n = 0, 1, ...\}$ can be modeled as a controlled Markov chain with transition probabilities equal to the product of individual state transitions, as in Equation (1), where b' is defined by Equation (2). Here f is the packet goodput (correctly received packets), l is the number of packet arrivals, and N_b is the buffer's capacity.

$$P(s'|s, a) = P^b(b'|[b, h], a)P^h(h'|h)P^x(x'|x, a) \quad (1)$$

$$b' = \min(b - f + l, N_b) \quad (2)$$

From Equation (2), it can be concluded that P^b depends on the goodput distribution P^f. Assuming independent packet losses, $P^f(f|BEP, z) = binomial(z, 1 - PLR)$, where $PLR = 1 - (1 - BEP)^L$ is the packet loss rate for a packet with size of L (bits).

A post-decision state (PDS), represented by $\widetilde{s} \triangleq (\widetilde{b}, \widetilde{h}, \widetilde{x}) \in \mathcal{S}$, denotes a state of the system after all known/controllable dynamics have occurred but before the unknown dynamics occur [14, 15, 27]. In our problem,

$$\widetilde{s}^n = ([b^n - f^n], h^n, x^{n+1}). \quad (3)$$

We can formulate our problem in terms of PDSs instead of conventional states by decomposing the transition $s \to s'$ into two parts: a known transition $s \to \widetilde{s}$ with cost $c_k(s, a)$ and transition probability $P_k(\widetilde{s}|s, a)$, and an unknown transition $\widetilde{s} \to s'$ with cost $c_u(\widetilde{s})$ and transition probability $P_u(s'|\widetilde{s})$. We can define two optimal value functions $V^*(s)$ and $\widetilde{V}^*(\widetilde{s})$ over the conventional states and PDSs, respectively. The two value functions are related by the following equations:

$$\widetilde{V}^*(\widetilde{s}) = c_u(\widetilde{s}) + \gamma \sum\nolimits_{s' \in \mathcal{S}} P_u(s'|\widetilde{s})V^*(s') \quad (4)$$

$$V^*(s) = \min_{a \in \mathcal{A}(s)} \left\{ c_k(s, a) + \sum\nolimits_{\widetilde{s} \in \mathcal{S}} P_k(\widetilde{s}|s, a)\widetilde{V}^*(\widetilde{s}) \right\}. \quad (5)$$

Knowing $\widetilde{V}^*(\widetilde{s})$, the optimal policy π^* can be found by taking the action in each state that minimizes the right-hand side of Equation (5). To solve the problem online, we use the PDS learning algorithm [15, 18, 27], which is a stochastic iterative algorithm. PDS learning takes the greedy action in each time slot and updates the value of the present state $\widetilde{s}^n$ by using a weighted average of (i) the current PDS value function estimate $\widetilde{V}(s^n)$, and (ii) a new sample estimate of the PDS value function based on the next state's estimated value as:

$$\widetilde{V}^{n+1}(\widetilde{s}^n) = (1-\alpha^n)\widetilde{V}^n(\widetilde{s}^n)+\alpha^n[c_u^n(\widetilde{s}^n)+\gamma V^n(s^{n+1})]. \quad (6)$$

Since the unknown system dynamics are not dependent on the action taken, using PDSs obviates the need for action exploration. Algorithm 1 presents the pseudo-code for the PDS learning algorithm using an adaptive learning rate $\alpha^n \in [0, 1]$, where ***action evaluation*** requires computing $\{c_k(s^n, a) + \sum_{\widetilde{s}} P^k(\widetilde{s}|s^n, a)\widetilde{V}^n(\widetilde{s})\}$ in Equations (7) and (8).

Algorithm 1 Post-Decision State Learning

1: **initialize** $\widetilde{V}^0(\widetilde{s}) = 0$ for all $\widetilde{s} \in \mathcal{S}$
2: **for** time slot $n = 0, 1, 2, \ldots$ **do**
3: Take the greedy action:

$$a^n = \arg\min_{a \in \mathcal{A}} \left\{ c_k(s^n, a) + \sum_{\widetilde{s}} P^k(\widetilde{s}|s^n, a)\widetilde{V}^n(\widetilde{s}) \right\} \quad (7)$$

4: Observe PDS $\widetilde{s}^n$, next state s^{n+1}, unknown cost c_u^n
5: Evaluate the state value function at time $n + 1$:

$$V^n(s^{n+1}) = \min_{a \in \mathcal{A}} \left\{ c_k(s^{n+1}, a) + \sum_{\widetilde{s}} P^k(\widetilde{s}|s^{n+1}, a)\widetilde{V}^n(\widetilde{s}) \right\} \quad (8)$$

6: Calculate $\widetilde{V}^{n+1}(\widetilde{s}^n)$ using Equation (6)
7: **end for**

B. Conventional Q-Learning

For the algorithmic comparison, we also briefly introduce Q-learning. The key step in Q-learning is performing an update at the end of every time slot according to the current experience tuple: (s^n, a^n, c^n, s^{n+1}). The update can be expressed as:

$$\begin{aligned} &Q^{n+1}(s^n, a^n) \leftarrow \\ &(1 - \alpha^n)Q^n(s^n, a^n) + \alpha^n[c^n + \gamma \min_{a' \in \mathcal{A}} Q^n(s^{n+1}, a')], \end{aligned} \quad (9)$$

where s^{n+1} is distributed based on the transition probability distribution $P(s^{n+1}|s^n, a^n)$; a' is the greedy action in time slot $n+1$; α^n represents the time-varying learning rate parameter; and $Q^0(s, a)$ can be initialized arbitrarily for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. In the literature, many researchers have explored various Q-learning based RL hardware accelerator structures for better performance and lower power consumption [28–31]. However, these hardware optimization techniques are not, at least directly, applicable to our PDS learning algorithm, as PDS based methods are uniquely optimized for emerging wireless IoT systems to reduce the convergence time. Therefore, it is

important to exploit dedicated hardware accelerators for the PDS based learning algorithms.

III. Proposed Hardware Architecture

Here, we present an optimized hardware accelerator for the action evaluation step to improve the efficiency and hence facilitate real-world deployment of next-generation RL techniques. The proposed hardware accelerator is mainly composed by two components: Known Cost (KC) block and State Value Expectation (SVE) block, as shown in Fig. 1. Specifically, we optimize the lookup table (green), tree structure (blue), and data selection (orange), according to the unique characteristics of the PDS based RL algorithm to speedup the computation and reduce the power consumption. We present the detailed design and optimization approaches below.

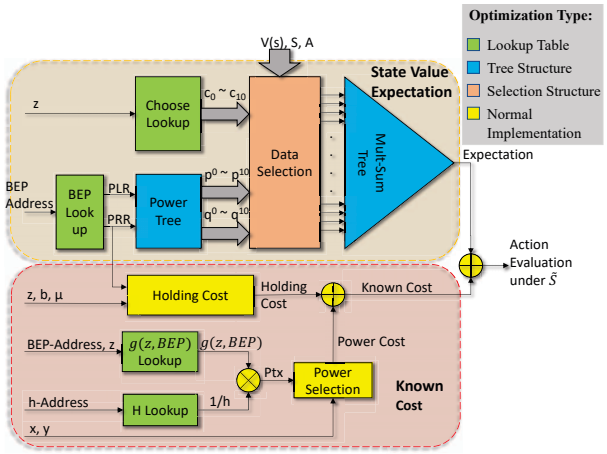


Fig. 1: Top-level architecture of the proposed hardware accelerator for action evaluation. It comprises two main blocks: Known Cost and State Value Expectation.

A. Lookup Table Reduction and State Encoding for RL

To avoid an infinite number of channel states in the proposed module, all analog states are quantized to discrete values. In order to further reduce the computational complexity, we design a lookup table reduction structure with state encoding. This reduces execution time and lowers the power consumption of the learning system, which are critical aspects for real-time wireless IoT systems [32].

At the beginning stages of our module, most computations are complex and computationally-intensive with heavy multiplications and power operations (e.g., $\binom{z}{i}$ when computing the Binomial goodput distribution, $PLR = 1 - PRR = 1 - (1 - BEP)^L$, and P_{tx} defined by Equation (10), where β is proportional to z, and erf() denotes the error function). However, the combinations of the inputs are limited by the size of state and action spaces. When the number of states is small, a lookup table is proved to be a promising choice for the implementation [33–35]. Therefore, we pre-process most computations at the input stage, which are then implemented as lookup tables, as shown in Fig. 1.

$$P_{tx} = \frac{\sqrt{2}N_0(2^{\beta}-1)\text{erf}^{-1}(1-\frac{\beta * BEP}{4})}{3*h}. \quad (10)$$

However, in the PDS learning algorithm, a large number of state values are not used after quantization (e.g. There are only 8 valid channel states, but 2^{32} possible inputs for a 32-bit system), which introduces redundant input space for the lookup table and negatively impacts the performance. To this end, the lookup tables for BEP and h are further optimized by state encoding. Discrete values are encoded into successive binary addresses to compress the input bit-width and unused cases, as shown in Fig. 2, which achieved a reduction of $61\times$ for unused case numbers. As a result, the circuit cost, speed, and power consumption are all improved by using a smaller input size. In our implementation, the bit-widths of both BEP and h are reduced from 32-bit to 3-bit for the 32-bit system. Furthermore, the encoded case input makes the circuit more re-programmable friendly across different applications [36, 37]. The inputs can be encoded similarly based upon the resolution used for the channel state and BEP (or any other continuous parameter), while the lookup tables can be easily updated for a different environment.

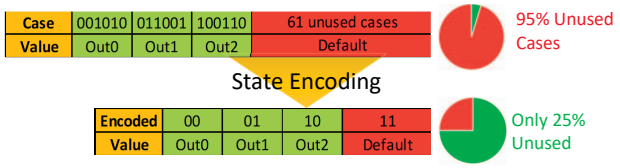
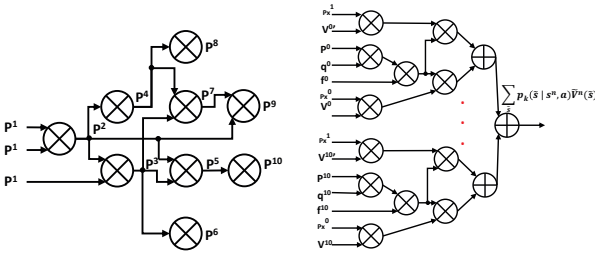


Fig. 2: An example of case encoding, where the input bit-width is compressed from 6 to 2, and unused cases are decreased over 61 times.

B. State Value Expectation (SVE)

Tree Structure: When calculating the SVE, all probabilities and state values for possible PDSs have to be collected and calculated (7), which makes the SVE block in general much slower than the KC block. Inspired by the parallel designs in recent works of efficient hardware implementation [38–40], we propose a parallelized structure for the SVE block with two tree structures: power tree (Fig. 3(a)) and multi-sum tree (Fig. 3(b)). The power tree takes a probability p as input and outputs all of the p^0-p^{10} simultaneously (all the outputs will be read out at the same time when the circuit finishes switching), while the multi-sum tree collects all PLR^i (packet loss rate), PRR^i (packet receive rate), $V(s)$, and chooses values $\binom{z}{i}$ based on the current state and action (77 values in total), then computes $\mathbb{E}(V(\tilde{s}))$ with only 3 multipliers and 5 adders. Besides accelerating the computation, the parallel design can also reduce power consumption since it decreases the critical path and eliminates the need for extra registers for data buffering or redundant computation.



(a) Power Tree (b) Multi-Sum Tree

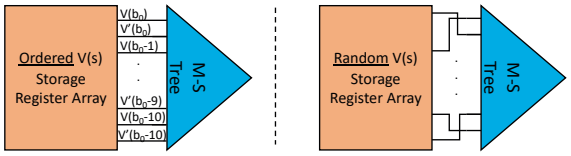
Fig. 3: The proposed parallel structures for (a) power tree and (b) multi-sum tree.

Data Selection: The data selection module in Fig. 1 is responsible for preparing the corresponding (z choose i) values and $V(s)$ for the multi-sum tree. For the considered learning model, there are 26 buffer states (b), 8 channel fading states (h), and 2 power management states (x), which leads to 416 different joint states in total. For each action, no more than 22 of those will be selected as the candidates for the PDS. In other words, the data selection module needs to select at most 22 $V(\widetilde{s})$ from all 416 $V(s)$. Thus, the complexity of this block is largely dependent on the addressing and indexing schemes for the data. To this end, ordered storage array is employed to reduce the circuit complexity, as shown in Fig. 4(a). It can be observed from Equation (3) that all possible PDSs $\widetilde{b}$ are continuous from the current index b to index $(b-z)$. By leveraging this property, we reorder the storage array in a way that all candidates of the PDSs for each possible case are stored consecutively. Consequently, only the position $V(s)$ for the current index b is required to locate, while the subsequent 21 $V(s)$ will be automatically addressed. Furthermore, since the channel state h does not vary across different PDSs ($\widetilde{h}=h$ as in Equation (3)), all $V(s)$ with the same h are indexed as a group. We also pre-select all $V(s)$ with the current h before reading the current action. All together, the $V(\widetilde{s})$ selection is optimized from 416-to-22 selection to 52-to-1 selection by using the proposed ordered storage array.

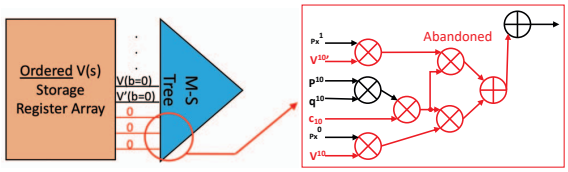
Another issue for this module is that the length of $\widetilde{s}$ changes based on the value of action z. We propose to adopt an auto-disable technique in the data selection module as shown in Fig. 4(b), which disables branches of the multi-sum tree when the number of possible buffer states for PDS ($\widetilde{b}$) is not at the maximum, i.e. $z < z_{max}$. Under this circumstance, the corresponding $V(\widetilde{s})$ and c_i for the unused branches are assigned as zeroes, which will automatically stop multipliers and adders of the corresponding branches from switching.

C. Known Cost (KC)

The computation of transmission power P_{tx} dominates the complexity of the KC block, which includes multiplications, power options, and the inverse error function, as expressed by Equation (10). To speed up the computation, we decompose



(a) Ordered storage array (left) vs. random storage array (right)



(b) Component auto-disable

Fig. 4: Hardware optimizations for data selection module.

$P_{tx} = g(z, BEP) * 1/h$, where $g(z, BEP)$ can be given by:

$$g(z, BEP) = \frac{\sqrt{2}N_0(2^{\beta}-1)\text{erf}^{-1}(1-\frac{\beta * BEP}{4})}{3}. \quad (11)$$

Consequently, we construct a lookup table for $g(z, BEP)$ of size $size(z) * size(BEP) = 10 \times 5$, which helps avoid integral and power computations.

IV. Experimental Results

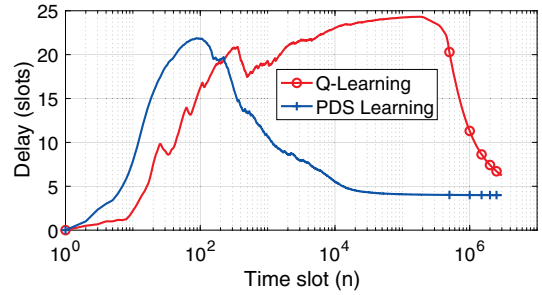
A. Learning Algorithm Comparison

Fig. 5 compares the simulated performance between our PDS learning implementation (Algorithm 1) and Q-learning. All results are generated by a MATLAB based simulator over 3,000,000 time slots. It can be seem from Fig. 5 that the PDS learning algorithm outperforms Q-learning in terms of both cumulative average delay and power consumption.

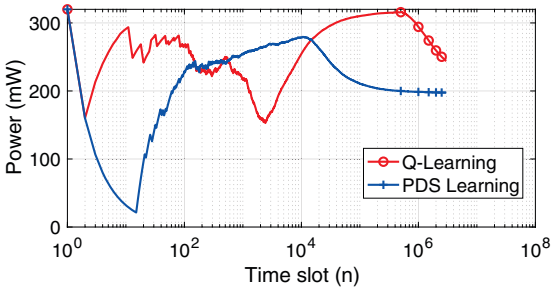
Besides power and delay, we further analyse the convergence speed of our algorithm in Fig. 6. The red curve (circle markers) denotes the cumulative average cost incurred up to time slot n by Q-learning (where the cost is defined in Section II-A as a weighted sum of the power cost and delay cost) and the blue curve (+ markers) denotes the cumulative average cost for PDS learning. While PDS learning approximately converges in 250,000 time slots, Q-learning has still not converged after 3,000,000 time slots, so it is at least 12 times slower than PDS learning. This shows that PDS learning is a better candidate for real-time IoT systems, where fast learning is needed to adapt to the real environment.

B. Hardware Implementation

We implemented and evaluated the following four approaches: Our proposed efficient action-evaluation architecture, a baseline straightforward hardware design without employing the proposed optimization, a software implementation with C++, and a Q-learning circuit using Verilog HDL. For a fair comparison, all common intrinsic variables and state values $V(s)$ use a bit-width of 32. They were all mapped to



(a) Cumulative average delay.



(b) Cumulative average power.

Fig. 5: Comparison between PDS learning and Q-learning.

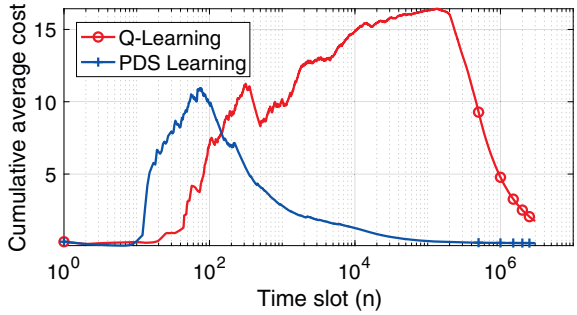


Fig. 6: Comparison of convergence speed.

a 32nm technology node using a Synopsys Design Compiler. The software is coded and tested with C++ on macOS, with 2.6 GHz 6-core Intel i7 processor and 16GB RAM. No multi-threaded optimization is added to the code, which means the software runs with only a single core under the limitation of macOS. As wireless IoT systems usually have less computing resources, we consider this setting as a guaranteed upper bound for the software implementation's speed.

We evaluate and compare the execution delays and average runtime for our two hardware designs and the software implementation of PDS learning. Furthermore, the power and area consumption of the optimized hardware accelerator and the baseline design are compared to illustrate the effectiveness of the proposed hardware optimization techniques. These results and comparisons are shown in Table I, where the execution times and power/area consumptions are also shown normalized to the optimized hardware design, for the baseline hardware design and software implementation. According to the experimental results, our optimized hardware accelerator is $3\times$ faster than the baseline circuit, while achieving a 49 times acceleration over the software implementation. The power consumption and cells number are also decreased by 85% and 86% respectively, compared to the baseline hardware design.

TABLE I: Optimized vs. Baseline Architectures (32-Bit)

	Optimized Hardware (PDS)	Baseline Hardware (PDS)	Software
Delay (ns)	86.97	258.31 ($3\times$)	4240 ($49\times$)
Power (mW)	6.17	41.21 ($7\times$)	-
# of Cells	93448	666543 ($7\times$)	-

The comparison between our proposed architecture for PDS learning and Q learning is presented in Table II. The implementation of Q-learning is based on Equation (9). According to the simulation results in Section IV-A, Q-learning converges over an order of magnitude slower than PDS based learning. Therefore, since the hardware will be activated once for each time slot, we normalize the hardware cost with respect to the convergence time for a fair comparison. These results show that the proposed PDS based learning accelerator achieves reductions of 83% and 59% in delay and power consumption, respectively, compared to Q-learning. Therefore, we can conclude that the proposed PDS learning architecture is faster and consumes less energy than Q-learning.

TABLE II: PDS vs. Q-learning on Hardware (32-Bit)

	Optimized Hardware (PDS)	Normalized Q-learning
Delay (ns)	86.97	521.9 ($6\times$)
Power (mW)	6.17	15 ($2.4\times$)

In addition, to achieve better performance according to the data range of a certain application scenario, designers vary the bit-width of the implementation [41, 42]. Thus, we also studied the hardware cost of our PDS learning accelerator for different bit-widths (i.e., 16, 32, and 64) as shown in Fig. 7. We normalize all the results to those for 16-bit. It can be observed that the hardware complexity increases approximately linearly with the increase of the bit-width.

V. Conclusion

We presented an efficient hardware accelerator for action evaluation of PDS based real-time RL for next generation wireless communication systems. By algorithmic and hardware co-optimization of the PDS learning implementation, we achieved a significant speedup for the action evaluation process of PDS, while simultaneously reducing its power consumption. Future work will be directed towards generalization of the proposed architecture to various wireless and IoT settings.

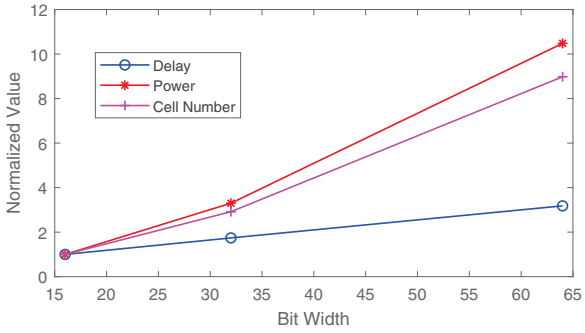


Fig. 7: Comparison of different bit-widths. All results are normalized to those for 16-bit, whose delay is 49.89 ns, power is 1.87 mW, and cell number is 32,030 cells.

REFERENCES

[1] J. Chakareski, "UAV-IoT for next generation virtual reality," *IEEE Trans. Image Process.*, vol. 28, no. 12, pp. 5977–5990, Dec. 2019.

[2] A. Seyedi and B. Sikdar, "Energy efficient transmission strategies for body sensor networks with energy harvesting," *IEEE Trans. Commun.*, vol. 58, no. 7, pp. 2116–2126, 2010.

[3] J. Chakareski and B. Girod, "Rate-distortion optimized packet scheduling and routing for media streaming with path diversity," in *Proc. IEEE Data Compression Conference*, Snowbird, UT, Mar. 2003, pp. 203–212.

[4] J. Chakareski, V. Velisavljević, and V. Stanković, "User-action-driven view and rate scalable multiview video coding," *IEEE Trans. Image Process.*, vol. 22, no. 9, pp. 3473–3484, Sep. 2013.

[5] N. Thomos, J. Chakareski, and P. Frossard, "Randomized network coding for UEP video delivery in overlay networks," in *Proc. IEEE Int'l Conf. Multimedia and Expo*, June/July 2009, pp. 730–733.

[6] J. Chakareski and P. Frossard, "Distributed collaboration for enhanced sender-driven video streaming," *IEEE Trans. Multimedia*, vol. 10, no. 5, pp. 858–870, Aug. 2008.

[7] N. Mastronarde, F. Verde, D. Darsena, A. Scaglione, and M. van der Schaar, "Transmitting important bits and sailing high radio waves: A decentralized cross-layer approach to cooperative video transmission," *IEEE J. Selected Areas in Communications*, vol. 30, no. 9, Oct. 2012.

[8] J. Chakareski, "Uplink scheduling of visual sensors: When view popularity matters," *IEEE Trans. Commun.*, vol. 2, no. 63, Feb. 2015.

[9] J. Chakareski, R. Aksu, X. Corbillon, G. Simon, and V. Swaminathan, "Viewport-driven rate-distortion optimized 360° video streaming," in *Proc. IEEE Int'l Conf. Communications*, May 2018.

[10] X. Corbillon, A. Devlic, G. Simon, and J. Chakareski, "Viewport-adaptive navigable 360-degree video delivery," in *Proc. IEEE Int'l Conf. Communications*. Paris, France: IEEE, May 2017.

[11] N. C. Luong, D. T. Hoang, P. Wang, D. Niyato, D. I. Kim, and Z. Han, "Data collection and wireless communication in Internet of Things (IoT) using economic analysis and pricing models: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, 2016.

[12] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (IoT) communication protocols," in *Proc. IEEE Int'l Conf. Information Technology (ICIT)*, 2017, pp. 685–690.

[13] F. Wu, C. Rüdiger, and M. Yuce, "Real-time performance of a self-powered environmental IoT sensor network system," *Sensors*, 2017.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 2018.

[15] N. Mastronarde and M. van der Schaar, "Joint physical-layer and system-level power management for delay-sensitive wireless communications," *IEEE Trans. Mobile Comput.*, vol. 12, no. 4, pp. 694–709, 2013.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[18] N. Mastronarde and M. van der Schaar, "Fast reinforcement learning for energy-efficient wireless communication," *IEEE Trans. Signal Process.*, vol. 59, no. 12, pp. 6262–6266, 2011.

[17] P. Blasco, D. Gunduz, and M. Dohler, "A learning theoretic approach to energy harvesting communication system optimization," *IEEE Trans. Wireless Commun.*, vol. 12, no. 4, pp. 1872–1882, 2013.

[19] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007, vol. 703.

[20] N. Toorchi, J. Chakareski, and N. Mastronarde, "Fast and low-complexity reinforcement learning for delay-sensitive energy harvesting wireless visual sensing systems," in *Proc. IEEE Int'l Conf. Image Processing*, Phoenix, AZ, USA, Sep. 2016, pp. 1804–1808.

[21] N. Sharma, N. Mastronarde, and J. Chakareski, "Delay-sensitive energy-harvesting wireless sensors: Optimal scheduling, structural properties, and approximation analysis," *IEEE Trans. Communications*, vol. 68, no. 4, pp. 2509–2524, Apr. 2020.

[22] N. Sharma, N. Mastronarde, and J. Chakareski, "Accelerated structure-aware reinforcement learning for delay-sensitive energy harvesting wireless sensors," *IEEE Trans. Signal Processing*, vol. 68, no. 1, Dec. 2020.

[23] N. Mastronarde, J. Modares, C. Wu, and J. Chakareski, "Reinforcement learning for energy-efficient delay-sensitive CSMA/CA scheduling," in *Proc. IEEE Global Telecommunications Conf.*, Dec. 2016.

[24] P. Kamalinejad, C. Mahapatra, Z. Sheng, S. Mirabbasi, V. C. Leung, and Y. L. Guan, "Wireless energy harvesting for the Internet of Things," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 102–108, 2015.

[25] A. Polianytsia, O. Starkova, and K. Herasymenko, "Survey of hardware IoT platforms," in *Proc. IEEE Int'l Scientific-Practical Conf. Problmes of Infocommunications Science and Technology*, 2016, pp. 152–153.

[26] A. Sengupta and S. Kundu, "Guest editorial: Securing IoT hardware: threat models and reliable, low-power design solutions," *IEEE Trans. Very Large Scale Integration Systems*, vol. 25, no. 12, 2017.

[27] N. Salodkar, A. Bhorkar, A. Karandikar, and V. Borkar, "An on-line learning algorithm for energy efficient delay constrained scheduling over a fading channel," *IEEE J. Sel. Areas Commun.*, vol. 26, no. 4, 2008.

[28] P. R. Gankidi, "FPGA accelerator architecture for Q-learning and its applications in space exploration rovers," Ph.D. dissertation, Arizona State University, 2016.

[29] A. Amravati, S. B. Nasir, S. Thangadurai *et al.*, "A 55nm time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots," in *Proc. IEEE Int'l Solid-State Circuits Conference*, 2018, pp. 124–126.

[30] H. Huang, R. S. Khalid, W. Liu, and H. Yu, "Work-in-progress: A fast online sequential learning accelerator for IoT network intrusion detection," in *Proc. IEEE Int'l Conf. Hardware/Software Codesign and System Synthesis*, 2017, pp. 1–2.

[31] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: An overview," *IEEE Comm. Surveys & Tutorials*, 2017.

[32] H. Kopetz, "Internet of things," in *Real-time systems*. Springer, 2011.

[33] M. Imani, D. Peroni, and T. Rosing, "Nvalt: Nonvolatile approximate lookup table for GPU acceleration," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 14–17, 2017.

[34] M. S. Abdulnabi and H. Ahmed, "Design of efficient cyclic redundancy check-32 using FPGA," in *Proc. IEEE Int'l Conf. Computer, Control, Electrical, and Electronics Engineering*, 2018, pp. 1–5.

[35] Y. Tian, T. Wang, Q. Zhang, and Q. Xu, "ApproxLUT: A novel approximate lookup table-based accelerator," in *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design*, 2017, pp. 438–443.

[36] R. J. Francis, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays," in *Proc. ACM/IEEE Design Automation Conference*, 1991, pp. 613–619.

[37] Y. Takemura, "Lookup table and programmable logic device including lookup table," Feb. 14 2017, US Patent 9,571,103.

[38] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "DLAU: A scalable deep learning accelerator unit on FPGA," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[39] D. Kim, J. Ahn, and S. Yoo, "Zena: Zero-aware neural network accelerator," *IEEE Design & Test*, vol. 35, no. 1, pp. 39–46, 2017.

[40] W. Song, D. Koch, M. Luján, and J. Garside, "Parallel hardware merge sorter," in *Proc. IEEE Int'l Symp. Field-Programmable Custom Computing Machines*, 2016, pp. 95–102.

[41] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar, "Reducing power by optimizing the necessary precision/range of floating-point arithmetic," *IEEE Trans. Very Large Scale-Integration Systems*, 2000.

[42] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. IEEE Int'l Symp. High-Performance Computer Architecture*, 1999, pp. 13–22.