Hardware Acceleration for Postdecision State Reinforcement Learning in IoT Systems

Jianchi Sun, *Student Member, IEEE*, Nikhilesh Sharma, Jacob Chakareski, *Senior Member, IEEE*, Nicholas Mastronarde, *Senior Member, IEEE*, and Yingjie Lao, *Senior Member, IEEE*

***Abstract*—Reinforcement learning (RL) is increasingly being used to optimize resource-constrained wireless Internet of Things (IoT) devices. However, existing RL algorithms that are lightweight enough to be implemented on these devices, such as *Q*-learning, converge too slowly to effectively adapt to the experienced information source and channel dynamics, while deep RL algorithms are too complex to be implemented on these devices. By integrating basic models of the IoT system into the learning process, the so-called postdecision state (PDS)-based RL can achieve faster convergence speeds than these alternative approaches at lower complexity than deep RL; however, its complexity may still hinder the real-time and energy-efficient operations on IoT devices. In this article, we develop efficient hardware accelerators for PDS-based RL. We first develop an arithmetic hardware acceleration architecture and then propose a stochastic computing (SC)-based reconfigurable hardware architecture. By using simple bitwise computations enabled by SC, we eliminate costly multiplications involved in PDS learning, which simultaneously reduces the hardware area and power consumption. We show that the computational efficiency can be further improved by using extremely short stochastic representations without sacrificing learning performance. We demonstrate our proposed approach on a simulated wireless IoT sensor that must transmit delay-sensitive data over a fading channel while minimizing its energy consumption. Our experimental results show that our arithmetic accelerator is 5.3× faster than *Q*-learning and 2.6× faster than a baseline hardware architecture, while the proposed SC-based architecture further reduces the critical path of the arithmetic accelerator by 87.9%.**

***Index Terms*—Action evaluation, hardware acceleration, Internet of Things (IoT) systems, latency sensitive resource-constrained online operation, postdecision state learning, reinforcement learning, stochastic computing (SC), wireless communication.**

Manuscript received August 28, 2021; revised December 23, 2021 and February 14, 2022; accepted March 17, 2022. Date of publication March 30, 2022; date of current version June 7, 2022. The work of Jianchi Sun and Yingjie Lao was supported in part by the National Science Foundation (NSF) under Grant 2047384. The work of Nikhilesh Sharma and Nicholas Mastronarde was supported in part by the NSF under Grant ECCS-1711335. The work of Jacob Chakareski was supported in part by the NSF under Grant CCF-2031881, Grant ECCS-2032387, Grant CNS-2040088, Grant CNS-2032033, and Grant CNS-2106150; and in part by the Panasonic Chair of Sustainability at NJIT. *(Corresponding author: Yingjie Lao.)*

Jianchi Sun and Yingjie Lao are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634 USA (e-mail: jianchs@clemson.edu; ylao@clemson.edu).

Nikhilesh Sharma and Nicholas Mastronarde are with the Department of Electrical Engineering, University at Buffalo, Buffalo, NY 14260 USA (e-mail: nsharma9@buffalo.edu; nmastron@buffalo.edu).

Jacob Chakareski is with the Department of Informatics, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: jacob.chakareski@njit.edu).

Digital Object Identifier 10.1109/JIOT.2022.3163364

I. Introduction

EMERGING applications, such as autonomous driving, mobile augmented and virtual reality, remote multi-view sensing, personalized healthcare, virtual teleportation, unmanned aerial vehicles, 360° video streaming, remote robot navigation, cooperative video delivery, and telemetry [1]–[10], have been creating diverse capabilities for next-generation Internet of Things (IoT) systems. However, they are still bottlenecked by the limited capabilities and resources of IoT devices [11]–[13].

In many emerging wireless IoT systems, the captured latency-sensitive data and the channel dynamics are governed by stochastic processes that are unknown a priori. This introduces the necessity of a self-learning system that can dynamically adapt to such unknown dynamics and statistical information. To this end, reinforcement learning (RL) [14], [15] has proven to be a promising approach. For example, in recent studies, the well-known *Q*-learning algorithm [16] has been employed to maximize the throughput [17] of energy harvesting transmitters, to minimize the sum of data compression and transmission energy of energy harvesting transmitters [18], [19], and to optimally tradeoff power and delay in IoT edge computing [13], [20]. Although *Q*-learning is lightweight enough to be implemented on resource-constrained IoT devices, it converges too slowly to effectively adapt to the experienced information source and channel dynamics.

In parallel, deep RL has received increasing attention for its ability to solve difficult decision-making problems with large (and possibly continuous) state and action spaces, both from the machine learning community [21]–[25] and from the wireless networking community [26]–[28]. However, deep RL algorithms have complex deep neural network architectures that make them infeasible to implement on resource-constrained wireless IoT systems where power, memory, and computational resources are limited [29], [30].[1] Worse still, deep RL algorithms are typically trained offline; therefore, they are not suitable for real-time learning where both training and decision-making need to be performed online, at runtime. For these reasons, none of the previously cited papers [26]–[28] deploy deep RL algorithms directly on end devices and all of them train the algorithms offline. For

[1]For instance, in a recent study [31], even with optimizations to adapt deep neural networks to low-power spectrum sensing applications, their solution still required at least one 128-output hidden layer to achieve relatively good performance, and the training phase of their model had to be executed on a powerful GPU.

instance, [26] investigates buffer-aware video streaming in a small-cell wireless network, [27] studies uplink scheduling for multiple energy-harvesting user equipments in a small-cell IoT system, and [28] demonstrates scheduling control in sliced 5G networks through an open radio access network (O-RAN). All of these deploy the trained deep RL agent at the base station or in the RAN, where sufficient computational resources are available.

To address the limitations of the existing approaches described above, our prior work advanced the concept of *post-decision states* (PDS) [15], [32]–[36], as have others [12], [14], [37]. PDSs allow us to exploit basic system knowledge to improve the learning performance. Concretely, the learning problem is decomposed into *known* and *unknown* components, by identifying the transitory system state after the execution of an action (hence the name PDS) and prior to the unknown system dynamics taking place. With this property, PDS-based RL is capable of significantly accelerating the learning convergence rate compared to Q-learning, but this comes at the cost of additional computational complexity to integrate the known components into the algorithm. Although PDS learning is far less complex than deep RL, its complexity may still hinder its real-time implementation on resource-constrained IoT devices.

On the other hand, although software is a remarkable option in most use cases due to its great flexibility, recent literature demonstrates that hardware acceleration is essential for various machine learning methods to enable real-time and lightweight applications in resource-constrained wireless IoT systems [38]–[43]. Following this direction, this article exploits efficient hardware architectures for PDS learning. We first design a hardware accelerator for the action evaluation (AE) step of PDS learning, which evaluates the value of a prospective action. This was presented in our earlier short preliminary study [43].[2] Then, we propose a stochastic computing (SC)-based and reconfigurable hardware architecture for the PDS learning algorithm. Specifically, by adopting SC, we eliminate the costly multiplications involved in the AE and replace them with estimation from samples, which hence simultaneously reduces the hardware area and power consumption. Thanks to the resiliency of PDS learning to stochastic perturbations, we can further improve the computational efficiency by using extremely short stochastic representations (i.e., each signal is represented by a very small number of stochastic samples) without sacrificing arithmetic performance. To differentiate from the SC-based accelerator, we refer to the arithmetic accelerator as the arithmetic circuit in the rest of this article. The main contributions of this article are summarized as follows.

1) We extend our short preliminary study in [43] to design a hardware accelerator for PDS learning algorithms. The proposed arithmetic-based design is 5.3× faster than Q-learning while consuming 59% less power.
2) We propose a novel SC-based hardware architecture, referred to as the transition probability distribution estimator (TPDE), for calculating the known transition probability from the state to the PDS without using multipliers. Built upon our prior design in [43], TPDE further accelerates the required computation and reduces the induced power consumption.
3) Since the PDS learning algorithm is inherently robust to stochastic perturbations, we show that our proposed hardware architecture is capable of achieving good performance with a small number of samples for the stochastic representation. Hence, we are able to significantly reduce the number of clock cycles for each computation, as opposed to typical SC-based systems that suffer from either large latency or a large number of processing elements (PEs) in a parallel architecture.
4) We introduce a high degree of reconfigurability into the hardware accelerator. It can be adapted to any state and action configuration within the designed maximum capability, while exploiting the tradeoff between speed and energy consumption.
5) The advantages of the proposed hardware architecture are comprehensively verified by experimental results. We show that the proposed SC-based architecture further accelerates the computation of the state value expectation (SVE) by at least 8.3 times.

[2]An earlier version of this article was presented at the 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI).

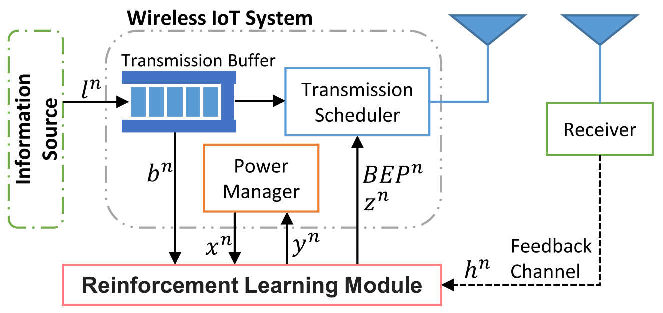


Fig. 1. Wireless IoT system model.

The remainder of this article is organized as follows. Section II introduces the system model that we use to illustrate the proposed approach, and reviews background on Q-learning, deep Q-learning (DQL), PDS-based RL, and SC. We describe our proposed architecture in detail in Section III and present our experimental results in Section IV. Finally, Section V concludes this article.

II. Background

A. System Model

We assume that a resource-constrained wireless IoT sensor must transmit delay-sensitive data over a fading channel to a receiver, while minimizing its power consumption. The system operates over discrete time steps indexed by $n \in \{0, 1, \ldots\}$, with fixed length ΔT seconds.

Fig. 1 illustrates the considered wireless IoT system. At the beginning of time step n, the RL module observes the system's state $s^n \triangleq (b^n, h^n, x^n) \in \mathcal{S}$, where $b^n \in \mathcal{S}_b = \{0, 1, \ldots, N_b\}$ is the finite buffer state, which represents the number of packets waiting in the buffer to be transmitted; $h^n \in \mathcal{S}_h$

is the channel state, which represents the discretized channel gain between the transmitter and receiver; $x^n \in \mathcal{S}_x$ is the binary power management state, which indicates if the radio is "on" and ready to transmit, or "off" in a power-saving state; and $\mathcal{S} = \mathcal{S}_b \times \mathcal{S}_h \times \mathcal{S}_x$ is the discrete and finite set of states. Subsequently, the RL module takes an action $a^n = (\mathrm{BEP}^n, y^n, z^n) \in \mathcal{A}$, where $\mathrm{BEP}^n \in \mathcal{A}_{\mathrm{BEP}}$ is the target maximum bit-error probability (BEP) at the receiver; $y^n \in \mathcal{A}_y$ is the binary power management action, which indicates whether to turn "on" or "off" the radio; $z^n \in \mathcal{A}_z$ is the packet throughput, which specifies the number of packets to transmit; and $\mathcal{A} = \mathcal{A}_{\mathrm{BEP}} \times \mathcal{A}_y \times \mathcal{A}_z$ is the discrete and finite set of actions. In our specific model implementation (see Section III), there are a total of 416 states and 110 actions, which is relatively complex for resource-constrained wireless IoT devices.

In the remainder of this section, we describe the channel, physical layer, transmission power, power management, transmission buffer, and traffic models in detail.

Channel Model: We consider a frequency nonselective block fading channel with channel gain $h^n \in \mathcal{S}_h$ in time step n. As in prior work [12], [17], [18], [35], [44], [45], we assume that the set of channel states $\mathcal{S}_h$ is discrete and finite, that the channel state h^n is known and constant in each time step, and that it evolves over time according to a discrete-time Markov chain with transition probability function $P^h(h'|h)$. We determine the discretized channel state by defining fixed thresholds $0 = \tau_0 < \tau_1 < \cdots < \tau_{N_h}$, where N_h denotes the number of channel states. Then, we define the discretized channel state to be h_k if the channel gain falls in the interval $[\tau_k, \tau_{k+1})$.

Physical Layer Model: We consider a single-carrier single-input single-output physical layer with a fixed symbol period of T_s seconds. The physical layer supports M modulation schemes that achieve data rates β^n/T_s bits/s, where $\beta^n \in \{\beta_1, \beta_2, \ldots, \beta_M\}$ and β_m is the number of bits per symbol used by the mth modulation scheme. Therefore, to transmit z^n packets of size L bits in ΔT seconds, we must have

$$\beta^n = \lceil z^n L T_s / \Delta T \rceil \text{ bits/symbol} \tag{1}$$

where $\lceil x \rceil$ denotes the ceiling operator, which rounds x up to the nearest integer. In time step n, the transmission scheduler module in Fig. 1 takes as input the maximum BEP BEP^n and the desired packet throughput z^n, and then selects the modulation scheme according to (1).

Transmission Power Model: Let $P_{tx}(h, \mathrm{BEP}, z)$ Watts denote the power required to transmit $z \in \mathcal{A}_z$ packets in channel state $h \in \mathcal{S}_h$ with maximum BEP $\mathrm{BEP} \in \mathcal{A}_{\mathrm{BEP}}$. The transmission power $P_{tx}(h, \mathrm{BEP}, z)$ depends on the physical layer modulation scheme and is typically: 1) convex increasing in the number of transmitted packets; 2) higher for lower bit-error probabilities; and 3) higher in worse channel states. These assumptions hold for typical modulation schemes, such as M-ary PSK and M-ary QAM [46, Table 6.1], and under information-theoretic bounds on the minimum power required for error-free communication [47]. Note that as in [44] and [35], we do not consider coding, but it can be introduced by appropriately modifying (1) and defining $P_{tx}(h, \mathrm{BEP}, z)$. In the rest of this article, we consider M-ary QAM for illustration; however, our learning algorithm and hardware accelerator can be modified to consider other modulation schemes and transmission power models. Under M-ary QAM, the transmission power can be expressed as follows [46, Table 6.1]:

$$P_{tx}(h, \mathrm{BEP}, z) = \frac{\sqrt{2} N_0 (2^\beta - 1) \mathrm{erf}^{-1}\left(1 - \frac{\beta \cdot \mathrm{BEP}}{4}\right)}{3h} \tag{2}$$

where N_0 denotes the noise power spectral density, $\mathrm{erf}^{-1}(\cdot)$ denotes the inverse error function, and β is the number of bits per symbol determined using (1).

Power Management Model: To trade power for delay, the wireless transmitter can be in one of two power management states, $\mathcal{S}_x = \{\text{on}, \text{off}\}$, and can be switched "on" and "off" using one of two power management actions, $\mathcal{A}_y = \{s\_\text{on}, s\_\text{off}\}$.[3] We let P_{on} and P_{off} Watts denote the power consumed by the wireless transmitter in the "on" and "off" states, respectively, and P_{tr} watts denote the power required to transition between the "on" and "off" states. We assume that $P_{\text{tr}} > P_{\text{on}} > P_{\text{off}} > 0$; therefore, there is a high cost for switching between the states, but less power is consumed in the "off" state than in the "on" state. Importantly, packets can only be transmitted if $x = \text{on}$ and $y = s\_\text{on}$; otherwise, $z = 0$.

The total *power cost* ρ incurred by taking action $a = (\mathrm{BEP}, y, z) \in \mathcal{A}$ in channel state $h \in \mathcal{S}_h$ and in power management state $x \in \mathcal{S}_x$ can be expressed as a sum of the *transmission power* and the *system power*: i.e.,

$$\rho([h, x], \mathrm{BEP}, y, x) = \begin{cases} P_{\text{on}} + P_{tx}(h, \mathrm{BEP}, z), & \text{if } x = \text{on}, y = s\_\text{on} \\ P_{\text{off}}, & \text{if } x = \text{off}, y = s\_\text{off} \\ P_{\text{tr}}, & \text{otherwise.} \end{cases} \tag{3}$$

As in prior work [48], we assume that the power management state x^n evolves over time according to a discrete-time controlled Markov chain with the following transition probability function:

$$P^x(x'|x, y = s\_\text{on}) = \begin{array}{c c} & \begin{array}{cc} \text{on} & \text{off} \end{array} \\ \begin{array}{c} \text{on} \\ \text{off} \end{array} & \begin{pmatrix} 1 & 0 \\ \theta & 1-\theta \end{pmatrix} \end{array} \tag{4}$$

$$P^x(x'|x, y = s\_\text{off}) = \begin{array}{c c} & \begin{array}{cc} \text{on} & \text{off} \end{array} \\ \begin{array}{c} \text{on} \\ \text{off} \end{array} & \begin{pmatrix} 1-\theta & \theta \\ 0 & 1 \end{pmatrix} \end{array} \tag{5}$$

where the row and column labels represent the current power management state x and the next power management state x', respectively, and $\theta \in (0, 1]$ denotes the probability of a successful power management transition (from "off" to "on" or from "on" to "off"). For simplicity of exposition, we assume that the power management state transition is deterministic, i.e., $\theta = 1$; however, our learning algorithm and hardware accelerator can be extended to the nondeterministic case.

Transmission Buffer and Traffic Model: At the end of the time step n, l^n new packets arrive into the IoT sensor's transmission buffer from the information source, where l^n is

[3]The power management action s_on should be interpreted as "stay on" in the "on" state or "switch on" in the "off" state; and s_off should be interpreted as "stay off" in the off state and "switch off" in the "on" state.

distributed according to the packet arrival distribution $P^l(l)$.[4] The buffer state evolves according to the following Lindley recursion:

$$b^{n+1} = \min\big(b^n - f^n\big(\text{BEP}^n, z^n\big) + l^n, N_b\big) \tag{6}$$

where N_b is the maximum number of packets that can be stored in the buffer and $f^n(\text{BEP}^n, z^n)$ is the packet goodput (i.e., the number of packets successfully delivered to the receiver). Note that $z^n \le b^n$ because it is not possible to transmit more packets than are in the buffer and $f^n(\text{BEP}^n, z^n) \le z^n$ because it is not possible to receive more packets than are transmitted. We assume that the value of f^n is sent to the transmitter over the feedback channel at the end of time step n.

Assuming that bit-errors are independent, the packet loss rate (PLR) can be expressed as

$$\text{PLR} = 1 - (1 - \text{BEP})^L \tag{7}$$

where L is the packet size in bits, and the goodput f has the following binomial distribution:

$$\begin{aligned} P^f(f|\text{BEP}, z) &= \text{Bin}(z, 1 - \text{PLR}) \\ &= \binom{z}{f}(1 - \text{PLR})^f(\text{PLR})^{z-f} \end{aligned} \tag{8}$$

where $\binom{z}{f} = z!/f!(z-f)!$. Importantly, since packets arrive at the end of each time step, packets that arrive in time step n cannot be transmitted until time step $n+1$ or later. Moreover, any packets that are not successfully delivered to the receiver in time step n remain in the buffer to be retransmitted in a future time step. Based on the above discussion, the buffer state b^n evolves over time according to a discrete-time controlled Markov chain with the following transition probability function:

$$\begin{aligned} &P^b\big(b'|b, \text{BEP}, z\big) \\ &\quad = \sum_{l=0}^{\infty}\sum_{f=0}^{z} P^f(f|\text{BEP}, z)P^l(l)\mathbb{I}_{\{b'=\min(b-f+l,N_b)\}} \end{aligned} \tag{9}$$

where $\mathbb{I}_{\{\cdot\}}$ is an indicator function that is set to 1 when the condition in $\{\cdot\}$ is true and is set to 0 otherwise.

Recall that our goal is to transmit *delay-sensitive* data while minimizing the IoT sensor's power consumption. We already defined the power cost in (3). Now, we need to define the expected *buffer cost*, which we introduce to penalize buffer delays and overflows. The expected buffer cost incurred when transmitting $z \in \mathcal{A}_z$ packets with target maximum BEP $\text{BEP} \in \mathcal{A}_{\text{BEP}}$ in buffer state $b \in \mathcal{S}_b$ can be expressed as

$$\begin{aligned} g(b, \text{BEP}, z) = &\sum_{l=0}^{\infty}\sum_{f=0}^{z} P^f(f|\text{BEP}, z)P^l(l) \\ &\times \{[b-f] + \eta \min(b-f+l-N_b, 0)\} \end{aligned} \tag{10}$$

where the *holding cost* $b-f$ penalizes large buffer states, the *overflow cost* $\eta \min(b-f+l-N_b, 0)$ penalizes each packet overflow by $\eta > 0$, and the expectation is taken with respect to the packet arrival distribution P^l and goodput distribution P^f.

[4]We assume that the arrivals in each time step are independent and identically distributed; however, the proposed system model can be extended to include Markovian traffic arrivals.

B. Markov Decision Process Formulation

The problem described above can be formulated as a *Markov decision process* (MDP) with discrete and finite state space $\mathcal{S} = \mathcal{S}_b \times \mathcal{S}_h \times \mathcal{S}_x$ and discrete and finite action space $\mathcal{A} = \mathcal{A}_{\text{BEP}} \times \mathcal{A}_y \times \mathcal{A}_z$. The state s^n evolves over time according to a discrete-time controlled Markov chain with *transition probability function*

$$P\big(s'|s, a\big) = P^b\big(b'|b, \text{BEP}, z\big)P^h\big(h'|h\big)P^x\big(x'|x, y\big) \tag{11}$$

and *cost function* defined as a weighted sum of the power and buffer costs: i.e.,

$$c(s, a) = \rho(s, a) + \lambda g(s, a) \tag{12}$$

where $\lambda \ge 0$ can be used to set the buffer cost constraint. The goal is to determine the optimal policy $\pi : \mathcal{S} \to \mathcal{A}$, which specifies the optimal action to take in each state to minimize the average power cost subject to an average buffer cost constraint.

For a given λ, the optimal solution satisfies the following Bellman equation:

$$V^*(s) = \min_{a\in\mathcal{A}} \underbrace{\left\{c(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s, a)V^*\big(s'\big)\right\}}_{Q^*(s,a)} \quad \forall s \in \mathcal{S} \tag{13}$$

where $V^*(s)$ is the *optimal value function*, which indicates how good it is to be in each state when following the optimal policy $\pi^*(s)$, and the related optimal *action-value function* $Q^*(s, a)$ indicates how good it is to take an arbitrary action in each state and then follow the optimal policy thereafter. The optimal policy $\pi^*(s)$ can be determined by taking the action that minimizes the right-hand side of (13) in each state.

If the cost and transition probability functions are known, then the optimal value function can be computed numerically using dynamic programming (e.g., value iteration or policy iteration [14]) and the optimal value of λ that satisfies the buffer cost constraint can be computed using the subgradient method. In the considered problem, however, the cost function in (12) is only partially known because the buffer cost in (10) depends on the unknown packet arrival distribution $P^l(l)$. Moreover, the transition probability function $P(s'|s, a)$ defined in (11) is only partially known because the buffer state transition probabilities $P^b(b'|b, \text{BEP}, z)$ defined in (9) depend on the unknown packet arrival distribution $P^l(l)$, and the channel state transition probabilities $P^h(h'|h)$ are unknown. Hence, the optimal value function and policy cannot be computed using dynamic programming; instead, they must be learned online, based on experience. Q-learning is a popular approach for this task, as described next.

C. Q-Learning

In each time step n, Q-learning updates an estimate of the action-value function based on the observed experience tuple (s^n, a^n, c^n, s^{n+1}), which comprises the current state, selected action, incurred cost, and next state. The update is performed as follows:

$$Q^{n+1}\left(s^n, a^n\right) \leftarrow \left(1-\alpha^n\right)Q^n(s^n, a^n) + \alpha^n\left[c^n + \gamma \min_{a'\in\mathcal{A}} Q^n\left(s^{n+1}, a'\right)\right] \tag{14}$$

where $s^{n+1} \sim P(\cdot|s^n, a^n)$ and $E[c^n] = c(s^n, a^n)$; a' is the greedy action in state s^{n+1}; $\alpha^n \in [0, 1]$ is a time-varying step size parameter; and $Q^0(s, a)$ can be initialized arbitrarily $\forall(s, a) \in \mathcal{S} \times \mathcal{A}$.

In the literature, many researchers have explored various Q-learning-based RL hardware accelerator structures for better performance and lower power consumption [20], [49], [50]. However, due to the limited training data and learning time for real-time learning, these hardware optimization techniques are not, at least directly, applicable in emerging wireless IoT systems because of Q-learning's slow convergence speed. In real-time learning, training data are generated or observed over time, which means that the agent has to wait for the new data no matter how fast each iteration is. Under these circumstances, slow convergence speed means that Q-learning will spend a relatively long period of time to reach the anticipated optimization level, during which energy and time are wasted. Different from Q-learning, PDS-based methods are uniquely optimized for the underlying wireless IoT system to increase the learning convergence speed.

D. Deep Q-Learning

Unlike tabular Q-Learning, DQL estimates action values with a deep Q-network (DQN [51]). By updating the weights of the DQN-based on minibatches of experience tuples, DQL learns successful policies directly from (possibly high-dimensional) sensory inputs and optimizes its action selection policy to fit the unknown dynamics.

In recent studies, DQL showed great potential in IoT wireless network optimization [26], [52]–[54]. Nevertheless, all their DQL agents run on powerful platforms, such as network servers, base stations, and satellites. Rajendran *et al.* [31] realized that deep learning was not suitable for low-power wireless applications and optimized their model, but it still required at least one hidden layer with 128 units to achieve relatively good performance, and only the inference phase could be performed on a low-power platform.

E. Postdecision State Learning

Before we can describe PDS learning, we need to formally introduce the PDS concept. A PDS denotes a state of the system after all known and controllable effects of the action have occurred but before the unknown dynamics occur [12], [14], [32]. In our wireless IoT system, the PDS in time step n is defined as follows:

$$\tilde{s}^n \triangleq \left(\tilde{b}^n, \tilde{h}^n, \tilde{x}^n\right) = \left(\left[b^n - f^n\right], h^n, y^n\right) \in \mathcal{S} \tag{15}$$

where $\tilde{b}^n = b^n - f^n$ denotes the buffer state after packets are successfully delivered to the receiver, but before new packets arrive;[5] $\tilde{h}^n = h^n$ since we do not know anything about the channel state transition; and $\tilde{x}^n = y^n$ since we assume that the power management state transition is deterministic. Given the PDS in time step n, we can express the state in time step $n+1$ as follows:

$$\begin{aligned} s^{n+1} &= \left(b^{n+1}, h^{n+1}, x^{n+1}\right) \\ &= \left(\min\left(\tilde{b}^n + l^n, N_b\right), h^{n+1}, \tilde{x}^n\right) \end{aligned} \tag{16}$$

where $l^n \sim P^l(\cdot)$ and $h^{n+1} \sim P^h(\cdot|\tilde{h}^n)$ denote the realizations of the packet arrivals and next channel state, respectively.

We formulate our problem in terms of PDSs by decomposing the transition $s \to s'$ into two parts: 1) a known transition $s \to \tilde{s}$ with expected cost $c_k(s, a)$ and transition probabilities $P_k(\tilde{s}|s, a)$ and 2) an unknown transition $\tilde{s} \to s'$ with expected cost $c_u(\tilde{s})$ and transition probabilities $P_u(s'|\tilde{s})$, such that

$$P(s'|s, a) = \sum\nolimits_{\tilde{s}} P_k(\tilde{s}|s, a)P_u(s'|\tilde{s}) \text{ and} \tag{17}$$

$$c(s, a) = c_k(s, a) + \sum\nolimits_{\tilde{s}} P_k(\tilde{s}|s, a)c_u(\tilde{s}). \tag{18}$$

Each of these factors can be easily derived based on the transition probability and cost functions defined in (11) and (12), respectively. For example, the unknown cost is nothing more than the expected overflow cost, i.e.,

$$c_u(\tilde{s}) = \eta \sum\nolimits_{l=0}^{\infty} P^l(l) \min\left(\tilde{b} + l - N_b, 0\right) \tag{19}$$

because the arrival distribution P^l is the only unknown component of the cost function defined in (12).

To map traditional RL to PDS learning, we define two value functions $V(s)$ and $\tilde{V}(\tilde{s})$ over the conventional states and PDSs, respectively. The corresponding optimal value functions are related by the following two Bellman equations:

$$\tilde{V}^*(\tilde{s}) = c_u(\tilde{s}) + \gamma \sum\nolimits_{s'\in\mathcal{S}} P_u(s'|\tilde{s})V^*\left(s'\right) \tag{20}$$

$$V^*(s) = \min_{a\in\mathcal{A}}\left\{c_k(s, a) + \sum\nolimits_{\tilde{s}\in\mathcal{S}} P_k(\tilde{s}|s, a)\tilde{V}^*(\tilde{s})\right\}. \tag{21}$$

Given the PDS value function $\tilde{V}^*(\tilde{s})$, the optimal policy $\pi^*(s)$ can be found by taking the action in each state that minimizes the right-hand side of (21).

To solve the problem online, we use the PDS learning algorithm presented in Algorithm 1 [15], [32]. First, the PDS value function $\tilde{V}^0(\tilde{s})$ is initialized to 0 for all $\tilde{s} \in \mathcal{S}$ (line 1). In each time step n, PDS learning takes the greedy action defined in (23) using the known cost (KC) function $c_k(s, a)$, the known transition probability function $P_k(\tilde{s}|s, a)$, and the current estimate of the PDS value function $\tilde{V}^n(\tilde{s})$ (line 3). Subsequently, PDS learning updates the estimated PDS value function as in (24) based on the observed experience tuple $(\tilde{s}^n, c_u^n, s^{n+1})$ (lines 4 and 5), where the PDS $\tilde{s}^n \sim P_k(\cdot|s^n, a^n)$ is defined in (15); the realization of the unknown cost

$$c_u^n = \eta \min\left(\tilde{b}^n + l^n - N_b, 0\right)$$

satisfies $E[c_u^n] = c_u(\tilde{s}^n)$, where $c_u(\tilde{s}^n)$ is defined in (19); and the next state $s^{n+1} \sim P_u(\cdot|\tilde{s}^n)$ is defined in (16). In [35], we proved that the sequence of PDS value functions $\tilde{V}^n$ generated by the PDS learning algorithm converges to $\tilde{V}^*$ with probability 1 as $n \to \infty$.

PDS learning has several advantages over Q-learning. First, only the unknown information in the transition $\tilde{s} \to s'$ needs

[5] Although we do not know the realization of the goodput f^n until the end of time step n, we know the goodput distribution defined in (8). This is sufficient to include f^n in the definition of the postdecision buffer state.

Algorithm 1 PDS Learning

```
1: initialize $\tilde{V}^0(\tilde{s}) = 0$ for all $\tilde{s} \in \mathcal{S}$
2: for time slot $n = 0, 1, 2, \dots$ do
3:     Take the greedy action:
           $a^n = \arg\min_{a \in \mathcal{A}} \left\{ c_k(s^n, a) + \sum_{\tilde{s}} P_k(\tilde{s}|s^n, a) \tilde{V}^n(\tilde{s}) \right\}$    (23)
4:     Observe PDS $\tilde{s}^n$, cost $c_u^n$, and next state $s^{n+1}$.
5:     Update $\tilde{V}^{n+1}(\tilde{s}^n)$:
           $\tilde{V}^{n+1}(\tilde{s}^n) = (1 - \alpha^n)\tilde{V}^n(\tilde{s}^n) + \alpha^n[c_u^n + \gamma V^n(s^{n+1})]$,    (24)
       where
           $V^n(s^{n+1}) = \min_{a \in \mathcal{A}} \left\{ c_k(s^{n+1}, a) + \sum_{\tilde{s}} P_k(\tilde{s}|s^{n+1}, a) \tilde{V}^n(\tilde{s}) \right\}$
6: end for
```

to be learned. Second, by updating the value of one PDS, we learn about all state–action pairs that can precede it due to the expectation over the known transition probabilities in both (23) and (24). Third, in RL, there is a tradeoff between *exploiting* actions that currently have the best estimated value and *exploring* other actions that might be better. However, if the unknown transition probabilities do not depend on the action (as in the considered problem), then PDS learning does not require exploration.

Together, the above three features significantly increase PDS learning's convergence speed compared to Q-learning; however, this comes at the cost of increased *action selection* and *learning update* complexity. In Q-learning, the action selection and update steps both require optimizing $Q^n(s, a)$ over the actions, so they have complexity $O(\mathcal{A})$. In PDS learning, in addition to optimizing over the actions, both (23) and (24) require calculating the action-value estimate $Q^n(s, a)$ for each prospective action based on the KC and transition probability functions.[6]

$$Q^n(s, a) = c_k(s, a) + \sum_{\tilde{s}} P_k(\tilde{s}|s, a)\tilde{V}(\tilde{s}). \tag{22}$$

Therefore, both steps have complexity $O(\mathcal{S} \times \mathcal{A})$. We will refer to the calculation in (22) as the *action evaluation* step. In Section III, we present efficient methods to calculate the *KC* $c_k(s, a)$ and the SVE $\sum_{\tilde{s}} P_k(\tilde{s}|s, a)\tilde{V}(\tilde{s})$, which appear in the AE step.

F. Stochastic Computing

To further optimize our hardware circuit, we design a TPDE based on SC. SC [55] enables complex computations to be performed using simple bitwise operations on streams of random bits. SC has recently been exploited for various low-energy or low-area applications, such as neural networks acceleration and 5G decoding [56]–[59]. In particular, SC is highly suitable for error-tolerant applications where approximated results are acceptable or certain errors in the intermediate stages are not perceivable by the end used [60], [61]. Moreover, SC enables very lightweight hardware implementations for resource-constraint devices. One example of an SC circuit is shown in Fig. 2(a). It can be seen that stochastic multiplication can be easily realized by an AND gate on the two bit-streams, as the probability to get a "1" as the output equals to the product of the equivalent probabilities for each of the inputs. In a typical SC architecture, stochastic number generators (SNGs) and comparators are also needed to convert binary signals to stochastic representations and stochastic bit-streams back to binary signals, respectively. To this end, a linear feedback shift register (LFSR) has been widely used as the SNG to generate stochastic bit-streams, as shown in Fig. 2(b), while a counter can effectively perform the stochastic-to-binary conversion, as illustrated in Fig. 2(c). Note that the goal of adopting SC is to accelerate the hardware computation, which is qualitatively different from Bayesian-based methods.

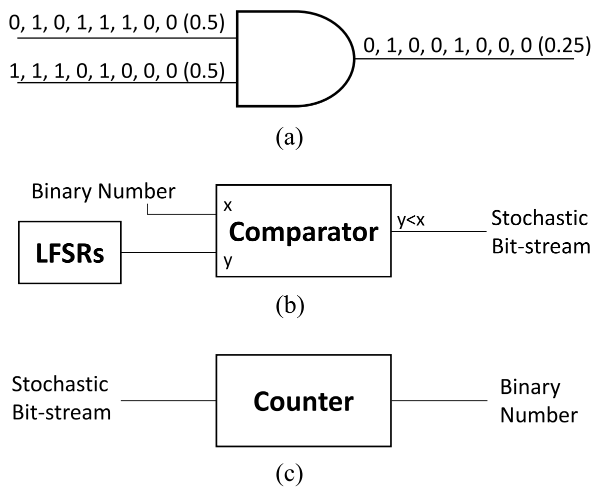


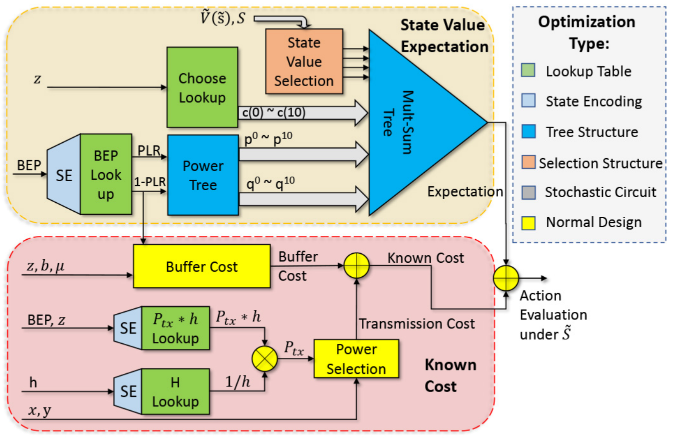
Fig. 2. SC circuit. (a) Stochastic multiplier implemented as an AND gate. (b) Stochastic bit-stream generator. (c) Stochastic-to-binary conversion.

Although SC offers simpler hardware for complex operations, it requires a long sequence of stochastic bits to obtain a precise result [56]. As a result, stochastic systems suffer from high latency or require a large number of PEs (e.g., AND gates for multiplication) to operate on the bit-streams in parallel. Thus, it is imperative to exploit ways for reducing the length of the bit-streams while maintaining the arithmetic performance. In Section III-C, we develop an SC-based accelerator to efficiently estimate the known transition probability function $P_k(\tilde{s}|s, a)$ rather than computing it arithmetically.

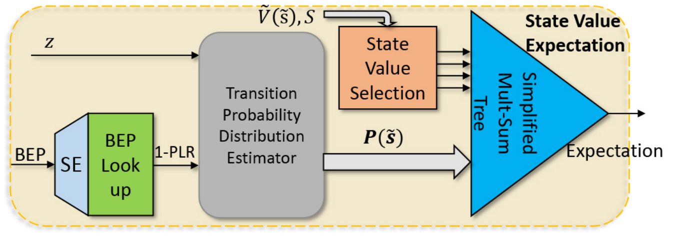
III. Proposed Hardware Architecture

To address the high computational complexity of PDS learning, we design an optimized hardware accelerator framework for the critical AE step in (22). As noted earlier, this step is performed once for each prospective action in both the action selection step (23) and the learning update step (24). For our accelerator framework, it consists of two main components: 1) the *KC block* for computing $c_k(s, a)$ and 2) the *SVE block* for computing $\sum_{\tilde{s}} P_k(\tilde{s}|s, a)\tilde{V}(\tilde{s})$. To realize a hardware accelerator for a specific system, we design the programmable

[6]PDS learning's action selection and update steps are given in (23) and (24), respectively, and require calculating $Q^n(s^n, a)$ and $Q^n(s^{n+1}, a)$, respectively, using (22) for each prospective action.



(a)



(b)

Fig. 3. AE hardware accelerator designs for the example system model. The SVE block is illustrated in (a) assuming that up to ten packets can be transmitted in each time step, i.e., $\mathcal{S}_z = \{1, 2, \ldots, 10\}$. (b) Alternative SVE module with TPDE.

lookup table (PLUT) (green) with state encoding (light blue), TPDE (gray), state value array (orange), and tree structure (blue), according to the unique characteristics of both the system and the PDS-based RL algorithm.

For illustration, in the remainder of this article, we consider an instance of the example system model in Section II-A with 26 buffer states ($b \in \mathcal{S}_b = \{0, 1, \ldots, 25\}$ packets), eight channel states ($h \in \mathcal{S}_h = \{-18.82, -13.79, -11.23, -9.37, -7.80, -6.30, -4.98, -2.08\}$ dB), two power management states ($x \in \mathcal{S}_x = \{\texttt{ON}, \texttt{OFF}\}$), two power management actions ($y \in \mathcal{A}_y = \{\texttt{SWITCH\_ON}, \texttt{SWITCH\_OFF}\}$), five target BEPs ($\text{BEP} \in \mathcal{A}_{\text{BEP}}$ yielding PLRs of 0.01, 0.02, 0.04, 0.08, and 0.16 for packets of size $L = 5000$ bits), and 11 transmission scheduling actions ($z \in \mathcal{S}_z = \{0, 1, \ldots, 10\}$). Therefore, there are a total of 416 system states and 110 possible actions. Although we consider this specific parameter configuration, the PDS learning algorithm and hardware acceleration architectures can be applied for any values of these parameters.

Fig. 3(a) illustrates an instance of the hardware accelerator design for the example system model in Section II-A, which is extended from our prior work [43]. Recall that we do not have complete information about our model because we do not know the data arrival probability distribution $P^l(l)$ or the channel state transition probabilities $P^h(h'|h)$. We briefly introduce the circuit functions below, while detailed circuit designing can be found in Sections III-A–III-C.

The bottom KC block in Fig. 3(a) calculates the known buffer cost and transmission cost, and then combines them to calculate the known components of (12). The known buffer cost only includes the known components of (10), which do not depend on $P^l(l)$, i.e.,

$$g_k(s, a) = \sum_{f=0}^{z} P^f(z|\text{BEP}, z)[b - f] \tag{25}$$

and is computed with an arithmetic circuit. For the transmission cost, the dominant part is the computation of P_{tx} defined in (2), where we implement two lookup tables to simplify the calculation. By multiplying P_{tx} by h, $P_{tx} * h$ lookup cancels the existence of h and stores the results for all the combinations of BEPs and zs. Then, with another lookup table outputting values for $1/h$, P_{tx} is calculated with very minimal cost.

The top block in Fig. 3(a) computes the SVE as

$$\sum_{\tilde{s} \in \mathcal{S}} P_k(\tilde{s}|s, a)\tilde{V}(\tilde{s})$$
$$= \sum_{\tilde{x} \in \mathcal{S}_x} \sum_{f=0}^{z} P^x(\tilde{x}|x, y)P^f(f|\text{BEP}, z)\tilde{V}(b - f, \tilde{x}, h) \tag{26}$$

where P^f is the goodput distribution defined in (8). The SVE block includes the following components.

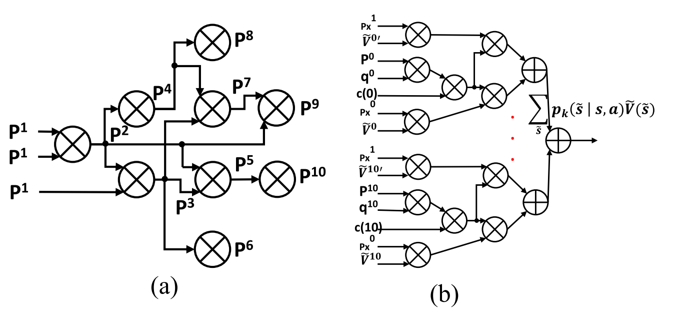
1) The BEP *Lookup* block takes as input the BEP's address and outputs both PLR and 1−PLR, where PLR is defined in (7).
2) The *Power Tree* block takes as input $p = \text{PLR}$ and $q = 1 - \text{PLR}$ and outputs $p^0, p^1, \ldots, p^{10}$ and $q^0, q^1, \ldots, q^{10}$, which are used to calculate the goodput distribution in (8).
3) The *Choose Lookup* block takes as input the transmission action z and outputs the values $c(f) = \binom{z}{f}$ when $f \leq z$ and $c(f) = 0$ when $f > z$, for $f = 0, 1, \ldots, 10$. The combinations $c(f)$ are also used to calculate the goodput distribution in (8).
4) The *State Value Selection* block takes as input the current state S and all state values, and then outputs the state values for possible PDSs.
5) Finally, the *Multisum Tree* block takes as input the outputs of the State Value Selection, Choose Lookup, and Power Tree blocks, and outputs the SVE.

More details about the Power Tree and State Value Selection blocks are provided in Section III-A.

Fig. 3(b) illustrates the proposed novel alternative SC-based SVE module, which we describe further in Section III-C.

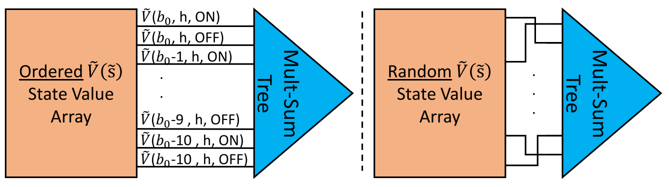
A. Tree Structure, Ordered State Value Array, and Component Autodisable

Tree Structure: The values of all possible PDSs and their corresponding probabilities are involved in the SVE calculation [see (23) and (24)]. This slows down the SVE module significantly and makes it critical to accelerating the circuit. Tracing this issue, we propose a parallelized structure for the SVE block that leverages two tree structures: 1) a *power tree* and 2) a *multisum tree* [blue blocks in Fig. 3(a)]. The



p¹
p²
p³
p⁴
p⁵
p⁶
p⁷
p⁸
p⁹
p¹⁰
(a)
$\sum_{\tilde{s}} p_k(\tilde{s} \mid s, a)\tilde{V}(\tilde{s})$
(b)

Fig. 4. Proposed parallel structures for (a) power tree and (b) multisum tree.



Ordered $\tilde{V}(\tilde{s})$ State Value Array
$\tilde{V}(b_0$, h, ON)
$\tilde{V}(b_0$, h, OFF)
$\tilde{V}(b_0$-1, h, ON)
$\tilde{V}(b_0$-9 , h, OFF)
$\tilde{V}(b_0$-10 , h, ON)
$\tilde{V}(b_0$-10 , h, OFF)
Mult-Sum Tree
Random $\tilde{V}(\tilde{s})$ State Value Array
Mult-Sum Tree

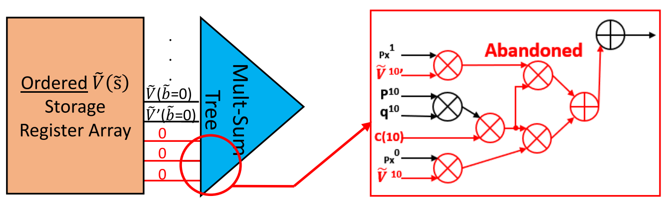
Fig. 5. Ordered storage array (left) versus random storage array (right).

power tree [Fig. 4(a)] takes a probability p as input and outputs the values $p^0, p^1, \ldots, p^{10}$ simultaneously. One instance of the power tree takes p = PLR as input and another takes $q = (1 - \text{PLR})$, and their outputs are used to calculate the goodput distribution in (8). Then, the multisum tree [Fig. 4(b)] takes the outputs of both power trees (p^i and q^i), the outputs of the Choose Lookup block ($c(0), \ldots, c(10)$), transition probability for $\tilde{x}$ (denoted by $P_{\tilde{x}}^{x'} = P(\tilde{x}|x, y)$ in any figures), and the corresponding PDS values (denoted by $\tilde{V}$ for $\tilde{x} = \texttt{ON}$ and $\tilde{V}'$ for $\tilde{x} = \texttt{OFF}$ in any figures). Then, it calculates the SVE according to (26) with only three stages of multipliers. These tree structures can accelerate the computation while reducing the power consumption since they decrease the critical path and eliminate the need for extra registers for data buffering or redundant computation.

Ordered State Value Array: During the AE step, a set of state values for each possible PDS needs to be selected among all the state values (i.e., $\tilde{V}(\tilde{s})$ for all PDS $\tilde{s}$ such that $P_k(\tilde{s}|s, a) \neq 0$). This process introduces two challenges to the hardware design: 1) the total number of states could change significantly based on the complexity of the system model and 2) the number of possible PDSs may vary, for instance, when the current buffer state b is smaller than the maximum value of the transmission action z in our example system model. We propose to use an ordered state value array and a component autodisable mechanism to simplify the computation.

In all cases, the range of possible PDSs is near the current state b_0, i.e., the PDS buffer state range $\{b_0 - z, b_0 - z + 1, \ldots, b_0\}$ in our system model is just like the area around a player's location in video game that can be reached within one step. Therefore, we reorder the storage array such that all candidates of the PDSs for each possible case are stored consecutively, as shown in Fig. 5. At the same time, we design the selection module to always output PDS values for $\tilde{b} = b_0$ to $(b_0 - z_{\max})$ for both $\tilde{x} = \texttt{ON}$ and $\texttt{OFF}$, since redundant state values will be canceled by the 0 s from the *Choose Lookup*. With all the designs above, the selection module needs to find only the location for $\tilde{V}(b_0)$ and then outputs it with its very next 21 state values. As a result, by implementing this for our wireless model, the selection module is reduced from 416-to-(2∼22) selection (total 416 states and possible 2∼22 PDSs) to 52-to-1 selection, which only finds b_0 (26-to-1) and x (2-to-1).

Component Autodisable With Visual State: In addition, the number of potential PDSs may vary depending on the current state, e.g., when a player moves to the edge of the map in a game and there are not many places to move; or the current buffer state is small and there are not many packets available to send in our system model. This brings challenges to both the selection module and multisum tree since they need to deal with different numbers of outputs and inputs. To avoid adding redundant control circuits, we add visual states that are out of the border of the state space but within the range of one action from the border [e.g., $\tilde{V}(\tilde{b} = -1)$ to $\tilde{V}(\tilde{b} = -10)$]. Those visual states have zeros as their state values so that the circuit can maintain the same output number for the selection module. The unused part of the multisum is disabled correspondingly based on (23). One example is illustrated in Fig. 6, where red paths are canceled by 0 s. Here, $\tilde{V}$ stands for $\tilde{x} = \texttt{ON}$ and $\tilde{V}'$ stands for $\tilde{x} = \texttt{OFF}$.



Ordered $\tilde{V}(\tilde{s})$ Storage Register Array
$\tilde{V}(\tilde{b}$=0)
$\tilde{V}'(\tilde{b}$=0)
0
0
0
Mult-Sum Tree
Abandoned
c(10)

Fig. 6. Component autodisable.

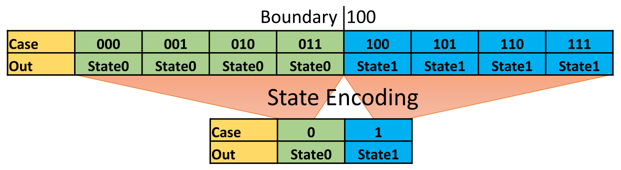


Fig. 7. Example of state encoding where the input bit-width is compressed from 3 to 1.

B. Programmable Lookup Table With State Encoding for RL

The channel state in the PDS learning algorithm is quantized into discrete state values. Since the number of states is typically limited to simplify the learning process and save energy in IoT applications, we implement lookup tables for the input stages to further accelerate the computation. For a direct implementation, there will be 2^{32} possible input values (for a 32-bit system) from the channel sensor, which corresponds to a 'costly' 32-bit lookup table. However, since many input cases share the same output and there are only eight channel fading states h in our model, we introduce state encoding (SE) to compress the input space of the lookup table. It encodes the input values into successive binary state addresses to compress the input bit-width, as illustrated in Fig. 7, where a 3-bit input

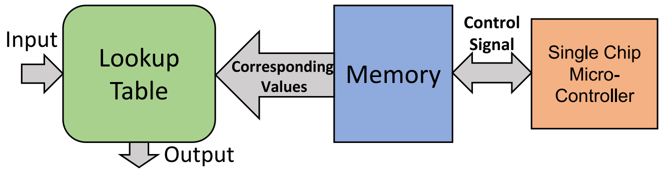


Fig. 8. PLUT with memory and SCM.

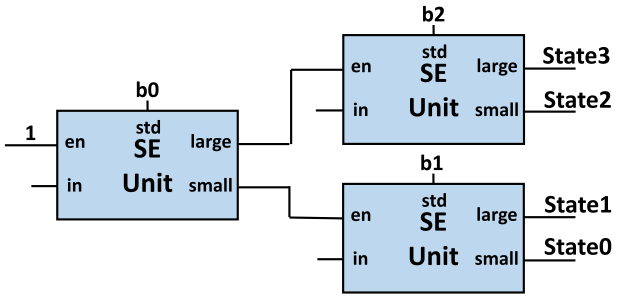


Fig. 9. Logic circuit of state encoding (SE) module (four states example).

is mapped into two states with "100" as the boundary. With state encoding applied, its input width is compressed by 3 (from 3 to 1 bit). Additionally, in order to adapt the same IoT circuit to various environments and use cases, the lookup table and state encoding are designed to be programmable with a memory module controlled by an SCM (single chip microcontroller). The functionality and state encoding of the lookup table are defined by the corresponding values from memory, which can be modified by the SCM, as shown in Fig. 8.

The circuit design for state encoding is shown in Fig. 9. Each block illustrates the basic SE unit, where port *in* takes the input value of the lookup table and *std* indicates the boundary value between the neighboring states that can be defined by the memory. The SE unit will compare *in* with *std* and then set one of the 1-bit outputs *large* or *small* to "1" and another to "0." Besides, when *en* is "0," both *large* and *small* will be set to "0", which can be simply implemented by logical AND operations.

By connecting multiple SE unit blocks as a binary tree structure and making all *in*s share the same input value as the input of the lookup table, we can easily obtain a programmable state encoding circuit for arbitrary state numbers. A four-state circuit design is demonstrated in Fig. 9, which has the function

$$\text{State} = \begin{cases} 0, \text{ if in } \in (0, b1] \\ 1, \text{ if in } \in [b1, b0) \\ 2, \text{ if in } \in [b0, b2) \\ 3, \text{ if in } \in [b2, +\infty). \end{cases} \tag{27}$$

The circuit for our lookup table is designed based on the SE unit, as shown in Fig. 10. S_0 to S_{n-1} are outputs of the state encoding circuit that correspond to n states. Then, the desired value can be quickly selected using AND gates, where D_0 to D_{n-1} are the corresponding output values from memory. To reconfigure the function for different use cases, we only need to update the boundary values and output values in the memory.

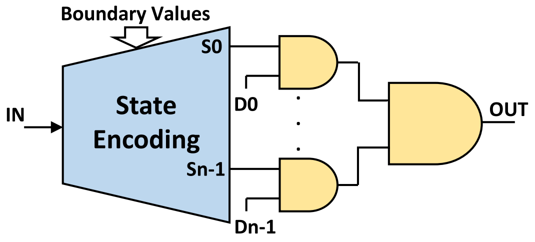


Fig. 10. Lookup table with state encoding.

C. Transition Probability Distribution Estimator and Stochastic Sample Generator

TPDE: The TPDE estimates the distribution of the PDS based on the current state and action as $P_k(\widetilde{s}|s, a)$, where $\widetilde{s}$ denotes the PDS and s and a are the current state and action, respectively. In PDS RL, this distribution is crucial as it needs to be computed at least two times in each time step (once for action selection and once for the learning update). However, calculating the entire transition probability distribution can be computationally expensive. For example, the transition probability distribution from the buffer state b to the post-decision buffer state $\widetilde{b} = b - f$ depends on the goodput distribution P^f defined in (8).

It can be seen that costly operations, including multiplications and powers, are involved in (8), which are not suitable for resource-constraint IoT systems. To tackle this challenge, we design a novel SC-based TPDE that can significantly reduce complexity while lowering power consumption. Based on the Monte Carlo sampling method, which is widely adopted for estimating expectations, in order to get

$$E[f(x)] = \sum\nolimits_x f(x)p(x) \tag{28}$$

we can sample L data points $\{x^1, \ldots, x^L\}$ and then establish an unbiased estimator for $E[f(x)]$

$$\hat{f} = \frac{1}{L}\sum\nolimits_{i=1}^{L} f(x^i). \tag{29}$$

The variance can be given by $var(\hat{f}) = (1/L)E[(f - E[f])^2]$, which indicates that the estimation accuracy improves with the sample size L. The goal of the TPDE is to estimate the transition probability distribution

$$P(S_i|S, A) = \sum\nolimits_{S'} f(S_i, S')P(S'|S, A) \tag{30}$$

where S_i is one specific case of the next state, $f(S_i, S') = 1$ when $S' = S_i$ and 0 when $S' \neq S_i$. By gathering L samples $S'_1, \ldots, S'_L$ for the PDSs from distribution $P(S'|S, A)$, based on (28) and (29), we can obtain $\hat{P}(S_i|S, A)$ as the unbiased estimator for $P(S_i|S, A)$, which is expressed as

$$\hat{P}(S_i|S, A) = \frac{1}{L}\sum\nolimits_{j=1}^{L} f(S_i, S'_j). \tag{31}$$

Thus, based on (30) and (31), we construct a TPDE with a sample generator $(P(S'|S, A))$ and a discriminator $(f(S_i, S'))$.

Stochastic Sample Generator (SSG): To obtain an accurate estimation for the transition probability distribution, it

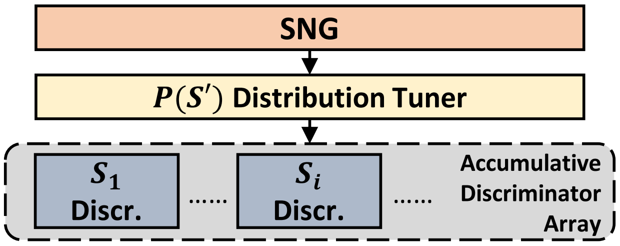


Fig. 11. Framework of the stochastic sample generator.

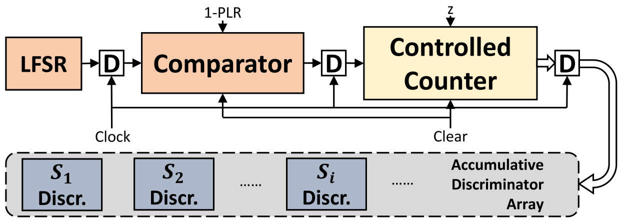


Fig. 12. TPDE for the binomial distribution family.

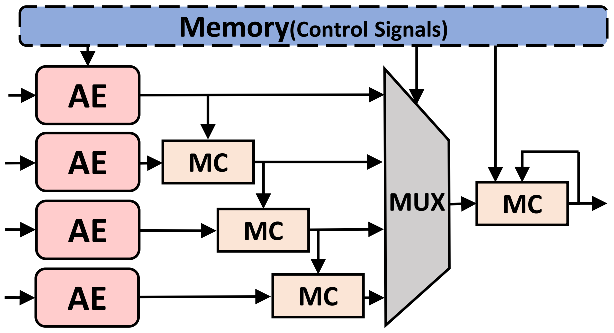


Fig. 13. Programmable 4-way AE structure.

is also crucial to design a sample generator that can generate samples based on the specific distribution. The design of our SSG is shown in Fig. 11, which consists of three main structures: 1) SNG; 2) distribution tuner; and 3) accumulative discriminator array.

We use the same SNG as in most prior SC designs, which is composed of LFSRs and a comparator that can generate a random bit-stream with a probability of P to be 1. After the SNG, the distribution tuner turns the bit-stream into samples based on the target distribution. For example, the tuner directly outputs each n bits as one sample for the binomial distribution in our PDS learning algorithm

$$S_i \sim \text{Bin}(P, n). \tag{32}$$

It is shown in prior works [62], [63] that the binomial distribution can be used to fit many other common distributions, such as Poisson distribution with $\lambda = nP$ and standard distribution with $\mu = nP$ and $\sigma^2 = nP(1-P)$. It is also possible to design a tuner for a logically descriptive distribution, similar to the distribution on the check node of the LDPC decoding [64].

Finally, the accumulative discriminator array will gather all the samples. Each discriminator will count the number of samples N_i that belong to the specific state S_i. The output of the S_i discriminator is an estimate of $L * P(S_i)$, i.e.,

$$P(S_i) \approx \frac{N_i}{L}. \tag{33}$$

Although a larger L will increase the accuracy of this estimation, we find that the PDS learning method implies remarkable tolerance to the random error, which means a small L can be adopted for acceleration and energy saving. This property is further discussed in Section IV-B.

TPDE Circuit Design for Binomial Distribution: The circuit design of the TPDE for the binomial distribution family ($\text{Bin}(n, p)$) is shown in Fig. 12, where the controlled counter is implemented as a distribution tuner. It takes a stochastic bit-stream and the throughput z (corresponding to the n of the binomial distribution), generates one sample for each z-bit, and informs the accumulative discriminator array when one sample is ready. For the accumulative discriminator array, each discriminator will count the number of received samples that belong to its state.

D. Programmable Parallel Greedy Action

In PDS-based RL, the AE step defined in (22) must be performed twice for every action in each time step (i.e., once for every action during the action selection step and once for every action during the learning update step). This presents challenges to wide applicability since the length of one time step can be small due to the high communication frequency, which brings the requirement of high-speed computation. On the other hand, in scenarios such as smart homes, saving energy becomes more important. Therefore, programmability is desired to enable a tradeoff between speed and power consumption for different applications. A 4-way example of the proposed programmable parallel structure is shown in Fig. 13. Here, AE represents the AE module as described above. MC is the minimum comparator module that takes two numbers as input and compares them, then outputs the smaller one. By connecting the MC module in series, we can then realize the arg min function. With the MUX gate at the output node, the parallelism can be configured by the control signals.

IV. Experimental Results

A. Experiments Setup

For software simulation, all algorithms are coded and tested with MATLAB on Windows 11, with a 3.80 GHz i7-10700K processor and 32-GB RAM. As wireless IoT systems usually have fewer computing resources, we consider this setting as a guaranteed upper bound for the software implementation's speed. For hardware testing, we implement our circuits with Verilog HDL, and then map them into a 32-nm technology node using Synopsys Design Compiler. All simulations are conducted using the state and action sets defined at the beginning of Section III and with packet size $L = 5000$ bits.

B. Algorithmic Performance

Fig. 14 compares the simulated performance of our PDS learning implementation (Algorithm 1), Q-learning, and DQL.

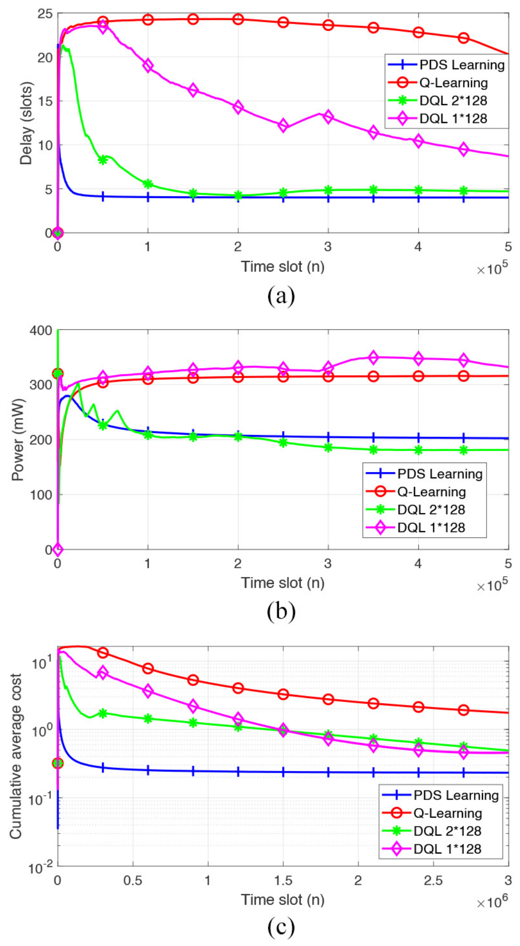


Fig. 14. Comparison between PDS learning, Q-learning, and DQL. (a) Cumulative average delay. (b) Cumulative average power. (c) Comparison of convergence speed.

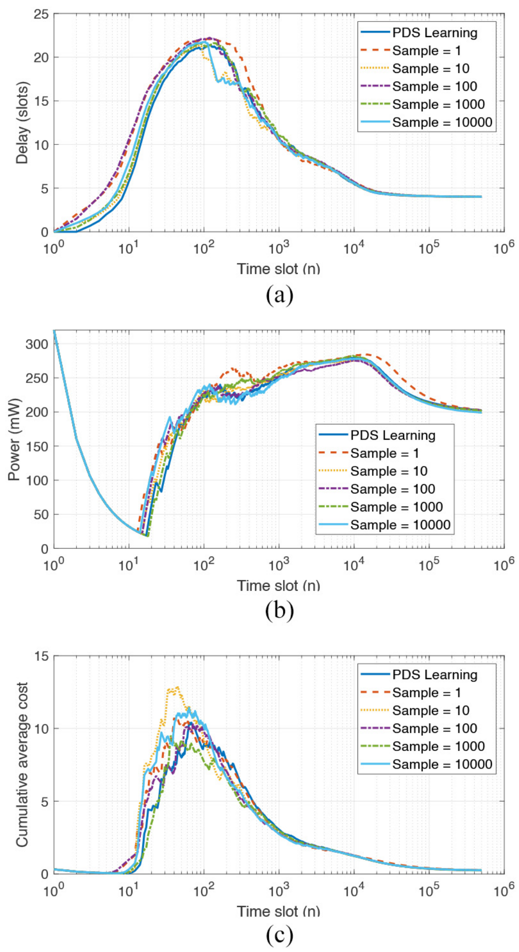


Fig. 15. Effect of stochastic process from SSG. (a) Cumulative average delay. (b) Cumulative average power. (c) Convergence speed.

DQL is implemented with MATLAB's deep RL toolbox. We examine two architectures with one and two fully connected hidden layers. The activation function is ReLU. The feature input layer for our model inputs current state, (b^n, h^n, x^n), to DQL and applies data normalization. The output layer is designed with the same size as the action space $\mathcal{A}$, so that each output corresponds to one possible action. In order to minimize the *cost function*, the reward for each action selection is defined as $-c(s, a)$. Consistent with the network size of a recent study [31] on low power wireless applications and the output layer's size for our model (110), we set the output size for each fully connected layer to be 128. The learning step size for DQL is 1×10^{-3}.

All results are averaged over at least 75 000 time slots. It can be seen from Fig. 14 that our PDS learning algorithm outperforms Q-learning and DQL in terms of both cumulative average delay and power consumption. Moreover, we find that DQL with one hidden layer (marked as "DQL 1×128") performs much worse than DQL with two hidden layers (marked as "DQL 2×128"), which further proves that DQL requires a relatively complex network in order to achieve acceptable performance.

We also evaluate the convergence speed of our algorithm in Fig. 14(c) with 3×10^6 time slots. 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 (12) as a weighted sum of the power cost and delay cost, which makes it the best representative of the overall performance) 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, and hence, is at least 12 times slower than PDS learning.

We now evaluate the algorithmic performance when using the TPDE. As discussed in Section III-C, the randomness introduced by the TPDE is highly dependent on the sample number L. By decreasing the sample number for each estimation, the delay and energy consumption of the TPDE can be reduced. However, the convergence of the learning algorithm may suffer from the estimator's high variance. To study the impact of this randomness on the learning process of our PDS model and to select the best sample number for the hardware test, we also evaluate the arithmetic performance of the SSG model. The same learning simulation processes are executed for sample numbers per estimation of a single PDS of 1, 10, 100, 1000, and 10 000. The results are shown as Fig. 15, which

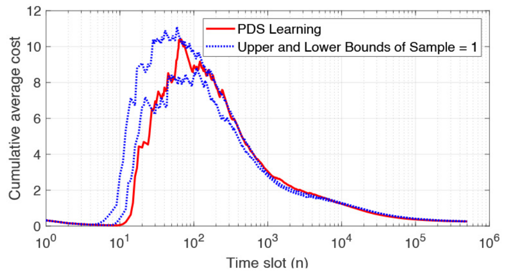


Fig. 16. Convergence for a single sample.

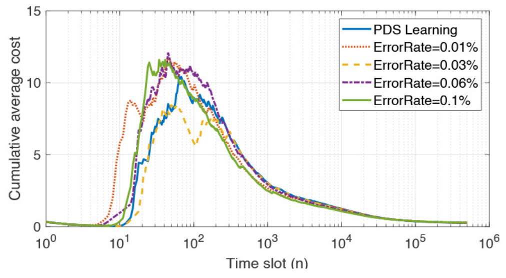


Fig. 17. Error tolerance.

show that all learning processes with different sample numbers converge similarly. Note that the differences between each curve are caused by the combination of the stochastic channel model, stochastic arrivals, and randomness from the TPDE. We further repeat the simulation of the learning process five times with only a single sample per estimation and compare the results with arithmetic PDS learning in Fig. 16, where we print the best and worst cumulative average cost among all five learning episodes for each time slot. It can be seen that all the learning curves have similar convergence speeds. Thus, we conclude that PDS learning is very resilient to the randomness introduced by SC, which can be leveraged to optimize the hardware cost by using a single sample without sacrificing the arithmetic performance.

C. Fault Tolerance

Fault tolerance is another advantage of SC, which indeed is also a desired characteristic for wireless IoT systems under noisy and low-energy environments. Many studies have shown that bit-flip errors are very common in those environments [65], while SC is inherently resilient to these soft transient errors [66]–[68]. Based on that we verify the error tolerance of our proposed method in Fig. 17, where we randomly flip the bits of all the outputs from multipliers in the power tree and multisum tree based on the error rate. The results show that our PDS learning accelerator achieves a high degree of error tolerance as all learning processes converge similarly.

D. Hardware Performance

We implement our proposed efficient architecture, a straightforward baseline design without employing the proposed optimization, and Q-learning using Verilog HDL. For a fair comparison, all common intrinsic variables and state values $V(s)$ use a bit-width of 32.

TABLE I
ARITHMETIC VERSUS BASELINE HARDWARE VERSUS Q-LEARNING (32-BIT)

	Arithmetic Hardware (PDS)	Baseline Hardware (PDS)	Normalized Q-learning	Software
Delay (ns)	98.76	258.31 (2.6×)	521.9 (5.3×)	1.04×10^6 $(10,531\times)$
Power (mW)	5.87	41.21 (7×)	15 (2.6×)	-
Area (# of cells)	92567	666543 (7.2×)	20040	-

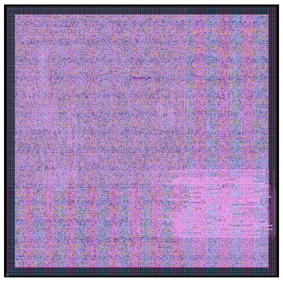


Fig. 18. Layout of the arithmetic hardware design.

We evaluate and compare the execution delays and average runtime for our two hardware designs and the software implementation of PDS learning. The power and area consumption of the arithmetic hardware accelerator and the baseline design is also 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 consumption are normalized with respect to those of the arithmetic hardware design. It can be observed that our arithmetic hardware accelerator is 2.6× faster than the baseline circuit while achieving a 1×10^4 times acceleration over the software implementation. Besides, the power and area consumptions are also decreased by 85.7% and 86.1%, respectively, compared to the baseline hardware design.

We use Synopsis IC compiler to generate the layout of the arithmetic hardware design with 32-nm technology, as shown in Fig. 18, where the postlayout area (not # of cells) and power are 0.38 mm^2 and 5.72 mW, respectively.

The implementation of Q-learning is based on (14). According to the simulation results in Section IV-B, Q-learning converges over an order of magnitude slower than PDS-based learning. We normalize the hardware cost with respect to the convergence time for a fair comparison. These results show that even though Q-learning costs less for a single iteration compared to PDS learning, when considering the convergence time, the proposed PDS-based learning accelerator yields reductions of 81% and 61% in delay and power consumption, respectively, compared to Q-learning. Therefore, we can conclude that the proposed PDS learning architecture achieves much superior hardware performance than Q-learning.

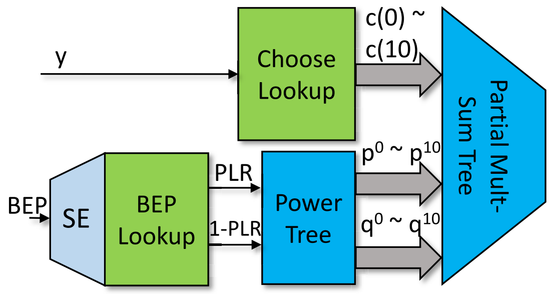


Fig. 19. Replaced circuit from the arithmetic accelerator.

TABLE II
COMPARISON WITH OUR PRIOR WORK [43] (32-BIT)

	Arithmetic Hardware (PDS)	TPDE
Delay (ns)	75.54	0.79
Power (uW)	3695	206
Area	132134	1095
Clk Freq (MHz)	12	1000
Latency	83ns	1-10ns
Power-delay Product	1×	.00067-.0067×

E. TPDE Versus Arithmetic Circuit

From the experimental results, we find that the delay of the Know Cost module is only 39.8% of the SVE module and the SVE module's delay takes 100% of the total delay (which means it is the critical path of the accelerator), indicating that the optimization for the SVE module is more crucial for speeding up the overall accelerator. This further confirms the motivation to adopt SC (i.e., TPDE) in the proposed architecture.

For a fair comparison, we implement TPDE and the corresponding circuit from the arithmetic accelerator (Fig. 19) that performs the same function as the TPDE. Here, the corresponding circuit is the SVE module without the state value selection module (as it is not included in the critical path) or adders at the output stage that perform the sum function. Both circuits are individually implemented under the same 32-bit input setting. The comparison of the arithmetic hardware architecture in our prior work [43] and the proposed TPDE is summarized in Table II, where the time per result for TPDE is defined by $([z \times \mathit{SampleNumber}]/\mathit{ClkFreq})$ $(z \in [1, 10])$. We set the sample number for one estimation as 1. It can be seen that the TPDE is 86.7% faster while consuming only 0.74% energy compared to the optimized arithmetic hardware architecture even with the largest packet throughput z.

From the results, we can see that the TPDE significantly reduces the energy consumption and circuit area as most stochastic circuits do. Besides that, the TPDE is 8.3× faster compared to the corresponding arithmetic circuit that executes the same function thanks to the resiliency of the PDS learning algorithm to the stochastic errors as shown in Fig. 15.

TABLE III
4-WAY PARALLEL AE (32-BIT)

	Non-Parallel	4-Way Parallel
Delay (ns)	98.76	102.45
Power (mW)	5.87	$6.04 * 4$

F. Programmable Parallel Greedy Action

To adapt our learning accelerator to broader application scenarios, we introduce programmable parallel greedy action in Section III-D. The comparison of nonparallel and 4-way parallel AE (Fig. 13) is shown in Table III. In the worst case (i.e., all four paths are activated), the additional MC modules and 4-to-1 MUX only incur an additional delay of 3.69 ns and 0.17 mW extra power consumption, which correspond to only 3.7% and 2.9% overhead, respectively.

V. CONCLUSION

This article presented efficient hardware architectures for accelerating PDS learning in IoT applications. We first designed a hardware accelerator for the most costly computation, i.e., the AE step. Then, building upon this architecture, we developed an SC-based hardware architecture, which can further simplify the computation while simultaneously reducing the power consumption. The effectiveness of the proposed methods is comprehensively verified from both arithmetic and hardware perspectives. Future work will be directed toward the generalization of the proposed architecture to various wireless and IoT settings.

REFERENCES

[1] J. Chakareski, "UAV-IoT for next generation virtual reality," *IEEE Trans. Image Process.*, vol. 28, pp. 5977–5990, 2019.
[2] J. Levinson *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2011, pp. 163–168.
[3] N. Thomos, J. Chakareski, and P. Frossard, "Randomized network coding for UEP video delivery in overlay networks," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jun./Jul. 2009, pp. 730–733.
[4] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.
[5] 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.
[6] 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. Sel. Areas Commun.*, vol. 30, no. 9, pp. 1597–1604, Oct. 2012.
[7] B. Ding, J. Kulkarni, and S. Yekhanin, "Collecting telemetry data privately," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 3574–3583.
[8] J. Chakareski, R. Aksu, X. Corbillon, G. Simon, and V. Swaminathan, "Viewport-driven rate-distortion optimized 360° video streaming," in *Proc. IEEE Int. Conf. Commun.*, Kansas City, MO, USA, May 2018, pp. 1–7.
[9] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.
[10] Y. He, C. Liang, F. R. Yu, and Z. Han, "Trust-based social networks with computing, caching and communications: A deep reinforcement learning approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 66–79, Jan.–Mar. 2020.

[11] K. Georgiou, S. Xavier-de Souza, and K. Eder, "The IoT energy challenge: A software perspective," *IEEE Embedded Syst. Lett.*, vol. 10, no. 3, pp. 53–56, Sep. 2018.

[12] N. Salodkar, A. Bhorkar, A. Karandikar, and V. S. Borkar, "An online learning algorithm for energy efficient delay constrained scheduling over a fading channel," *IEEE J. Sel. Areas Commun.*, vol. 26, no. 4, pp. 732–742, May 2008.

[13] X. Liu, Z. Qin, and Y. Gao, "Resource allocation for edge computing in IoT networks via reinforcement learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–6.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: 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, Apr. 2013.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[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, Apr. 2013.

[18] V. Hakami, S. Mostafavi, N. T. Javan, and Z. Rashidi, "An optimal policy for joint compression and transmission control in delay-constrained energy harvesting IoT devices," *Comput. Commun.*, vol. 160, pp. 554–566, Jul. 2020.

[19] M. Chincoli and A. Liotta, "Self-learning power control in wireless sensor networks," *Sensors*, vol. 18, no. 2, p. 375, 2018.

[20] Y. Debizet, G. Lallement, F. Abouzeid, P. Roche, and J.-L. Autran, "Q-learning-based adaptive power management for IoT system-on-chips with embedded power states," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2018, pp. 1–5.

[21] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[23] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[24] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, vol. 30. Red Hook, NY, USA: Curran, 2017.

[25] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[26] Y. Guo, F. R. Yu, J. An, K. Yang, Y. He, and V. C. M. Leung, "Buffer-aware streaming in small-scale wireless networks: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 7, pp. 6891–6902, Jul. 2019.

[27] M. Chu, H. Li, X. Liao, and S. Cui, "Reinforcement learning-based multiaccess control and battery prediction with energy harvesting in IoT systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2009–2020, Apr. 2019.

[28] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and learning in O-RAN for data-driven NextG cellular networks," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 21–27, Oct. 2021.

[29] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2018.

[30] J. Tang, D. Sun, S. Liu, and J.-L. Gaudiot, "Enabling deep learning on IoT devices," *Computer*, vol. 50, no. 10, pp. 92–96, 2017.

[31] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Deep learning models for wireless signal classification with distributed low-cost spectrum sensors," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 3, pp. 433–445, Sep. 2018.

[32] 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, Dec. 2011.

[33] 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. Conf. Image Process. (ICIP)*, 2016, pp. 1804–1808.

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

[35] N. Sharma, N. Mastronarde, and J. Chakareski, "Accelerated structure-aware reinforcement learning for delay-sensitive energy harvesting wireless sensors," *IEEE Trans. Signal Process.*, vol. 68, pp. 1409–1424, Feb. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8998306/

[36] N. Mastronarde, J. Modares, C. Wu, and J. Chakareski, "Reinforcement learning for energy-efficient delay-sensitive CSMA/CA scheduling," in *Proc. IEEE Global Telecommun. Conf.*, Dec. 2016, pp. 1–7.

[37] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, vol. 703. Hoboken, NJ, USA: Wiley, 2007.

[38] A. Singh, N. Chawla, J. H. Ko, M. Kar, and S. Mukhopadhyay, "Energy efficient and side-channel secure cryptographic hardware for IoT-edge nodes," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 421–434, Feb. 2019.

[39] S. Ebrahimi, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "Post-quantum cryptoprocessors optimized for edge and resource-constrained devices in IoT," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5500–5507, Jun. 2019.

[40] H. A. Gonzalez, S. Muzaffar, J. Yoo, and I. A. M. Elfadel, "An inference hardware accelerator for EEG-based emotion detection," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2020, pp. 1–5.

[41] L. Du *et al.*, "A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 198–208, Jan. 2018.

[42] A. Fayyazi, M. Ansari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "An ultra low-power memristive neuromorphic circuit for Internet of Things smart sensors," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1011–1022, Apr. 2018.

[43] J. Sun, N. Sharma, J. Chakareski, N. Mastronarde, and Y. Lao, "Action evaluation hardware accelerator for next-generation real-time reinforcement learning in emerging IoT systems," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 428–433.

[44] D. Zordan, T. Melodia, and M. Rossi, "On the design of temporal compression strategies for energy harvesting sensor networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 2, pp. 1336–1352, Feb. 2016.

[45] Q. Zhang and S. A. Kassam, "Finite-state Markov model for rayleigh fading channels," *IEEE Trans. Commun.*, vol. 47, no. 11, pp. 1688–1692, Nov. 1999.

[46] A. Goldsmith, *Wireless Communications*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[47] R. A. Berry and R. G. Gallager, "Communication over fading channels with delay constraints," *IEEE Trans. Inf. Theory*, vol. 48, no. 5, pp. 1135–1149, May 2002.

[48] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 6, pp. 813–833, Jun. 1999.

[49] S. Spanó *et al.*, "An efficient hardware implementation of reinforcement learning: The *Q*-learning algorithm," *IEEE Access*, vol. 7, pp. 186340–186351, 2019.

[50] L. M. D. Da Silva, M. F. Torquato, and M. A. C. Fernandes, "Parallel implementation of reinforcement learning Q-learning technique for FPGA," *IEEE Access*, vol. 7, pp. 2782–2798, 2019.

[51] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[52] C. Qiu, F. R. Yu, H. Yao, C. Jiang, F. Xu, and C. Zhao, "Blockchain-based software-defined industrial Internet of Things: A dueling deep *Q*-learning approach," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4627–4639, Jun. 2019.

[53] A. Asheralieva and D. Niyato, "Distributed dynamic resource management and pricing in the IoT systems with blockchain-as-a-service and UAV-enabled mobile edge computing," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 1974–1993, Mar. 2020.

[54] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao, "Service function chain embedding for NFV-enabled IoT based on deep reinforcement learning," *IEEE Commun. Mag.*, vol. 57, no. 11, pp. 102–108, Nov. 2019.

[55] B. R. Gaines, "Stochastic computing," in *Proc. Spring Joint Comput. Conf.*, 1967, pp. 149–156.

[56] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 8, pp. 1515–1531, Aug. 2018.

[57] A. Ren *et al.*, "SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 405–418, 2017.

[58] W. J. Gross, V. C. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *Proc. Conf. Rec. 39th Asilomar Conf. Signals Syst. Comput.*, 2005, pp. 713–717.

[59] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, "Delayed stochastic decoding of LDPC codes," *IEEE Trans. Signal Process.*, vol. 59, no. 11, pp. 5617–5626, Nov. 2011.
[60] B. K. Wong, T. A. Bodnovich, and Y. Selvi, "Neural network applications in business: A review and analysis of the literature (1988–1995)," *Decis. Support Syst.*, vol. 19, no. 4, pp. 301–320, 1997.
[61] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 1–19, 2013.
[62] L. H. Y. Chen, "On the convergence of poisson binomial to poisson distributions," *Ann. Probab.*, vol. 2, pp. 178–180, Feb. 1974.
[63] S. N. Stamenković, V. L. Marković, A. P. Jovanović, and M. N. Stankov, "Generalization of electron avalanche statistics based on negative binomial distribution—Multielectron initiation and Gaussian approximation," *J. Instrum.*, vol. 13, no. 12, 2018, Art. no. P12002.
[64] Y.-L. Ueng, C.-Y. Wang, and M.-R. Li, "An efficient combined bit-flipping and stochastic LDPC decoder using improved probability tracers," *IEEE Trans. Signal Process.*, vol. 65, no. 20, pp. 5368–5380, Oct. 2017.
[65] B. Sangchoolie, K. Pattabiraman, and J. Karlsson, "One bit is (not) enough: An empirical study of the impact of single and multiple bit-flip errors," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, 2017, pp. 97–108.
[66] B. Moons and M. Verhelst, "Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 4, no. 4, pp. 475–486, Dec. 2014.
[67] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *Proc. 52nd Annu. Des. Autom. Conf.*, 2015, pp. 1–3.
[68] J. Sartori, J. Sloan, and R. Kumar, "Stochastic computing: Embracing errors in architecture and design of processors and applications," in *Proc. 14th Int. Conf. Compilers Archit. Synth. Embedded Syst. (CASES)*, 2011, pp. 135–144.

Jianchi Sun (Student Member, IEEE) received the B.S. degree in electrical engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2015, and the M.S. degree in electrical engineering from Colorado State University, Fort Collins, CO, USA, in 2017. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, Clemson University, Clemson, SC, USA.

His research interests include digital VLSI design, hardware acceleration for machine learning, and artificial intelligence methodology.

Nikhilesh Sharma received the B.Tech. degree in electronics and communication engineering from the National Institute of Technology Srinagar, Srinagar, India, in 2014, and the M.S. and Ph.D. degrees in electrical engineering from University at Buffalo, Buffalo, NY, USA, in 2017 and 2020, respectively.

His research interests include deep learning, reinforcement learning, and Markov decision processes.

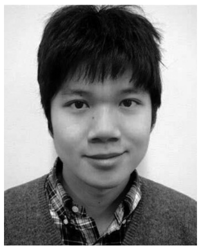
Jacob Chakareski (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Rice University, Houston, TX, USA, and Stanford University, Stanford, CA, USA.

He is an Associate Professor with the College of Computing, New Jersey Institute of Technology, Newark, NJ, USA, where he holds the Panasonic Chair of Sustainability and directs the Laboratory for AI-Enabled Virtual and Augmented Reality Immersive Communications and Networked Systems. He held research appointments with Microsoft Research, Redmond; HP Labs, Palo Alto, CA, USA; and EPFL, Lausanne, Switzerland, and served on the advisory board of Frame, Inc. His research has been supported by NSF, NIH, AFOSR, Adobe, Tencent Research, NVIDIA, and Microsoft. His research interests span next-generation virtual and augmented reality systems, UAV IoT sensing and networking, fast reinforcement learning, 5G wireless edge computing and caching, ubiquitous immersive communication, and societal applications.

Dr. Chakareski received the Adobe Data Science Faculty Research Award in 2017 and 2018, the Swiss NSF Career Award Ambizione in 2009, the AFOSR Faculty Fellowship in 2016 and 2017, and the Best Paper Awards at ICC 2017 and MMSys 2021. For further information, please visit www.jakov.org.

Nicholas Mastronarde (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the University of California at Davis, Davis, CA, USA, in 2005 and 2006, respectively, and the Ph.D. degree in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 2011.

He is currently an Associate Professor with the Department of Electrical Engineering, University at Buffalo, Buffalo, NY, USA. His research interests include reinforcement learning, Markov decision processes, resource allocation and scheduling in wireless networks and systems, UAV networks, and 5G and beyond networks.



Yingjie Lao (Senior Member, IEEE) received the B.S. degree from Zhejiang University, Hangzhou, China, in 2009, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Minnesota Twin Cities, Minneapolis, MN, USA, in 2015.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, USA.

Dr. Lao is the recipient of the NSF CAREER Award, the Best Paper Award at the International Symposium on Low-Power Electronics and Design, and the IEEE Circuits and Systems Society Very Large-Scale Integration Systems Best Paper Award. He is currently a member of four technical committees in the IEEE Circuits and Systems Society and IEEE Signal Processing Society.