IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 August 2024, accepted 17 September 2024, date of publication 20 September 2024, date of current version 3 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3464868

RESEARCH ARTICLE

Dynamic MAC Protocol for Wireless Spectrum Sharing via Hyperdimensional Self-Learning

YANG NI[1], (Graduate Student Member, IEEE), DANNY ABRAHAM[1], MARIAM ISSA[1], ALEJANDRO HERNÁNDEZ-CANO[2], MAHDI IMANI[3], (Senior Member, IEEE), PIETRO MERCATI[4], (Member, IEEE), AND MOHSEN IMANI[1], (Member, IEEE)

[1]Department of Computer Science, University of California at Irvine, Irvine, CA 92697, USA
[2]School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland
[3]Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA
[4]Intel Labs, Hillsboro, OR 97124, USA

Corresponding author: Mohsen Imani (m.imani@uci.edu)

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) Young Faculty Award, National Science Foundation under Grant 2127780, Grant 2319198, Grant 2321840, Grant 2312517, and Grant 2235472; in part by the Semiconductor Research Corporation (SRC), Office of Naval Research through the Young Investigator Program Award under Grant N00014-21-1-2225 and Grant N00014-22-1-2067; in part by the Air Force Office of Scientific Research under Grant FA9550-22-1-0253; and in part by Generous Gifts from Cisco.

ABSTRACT In wireless networks, dynamic spectrum access is the key to improving spectrum utilization and increasing channel capacity. Since the channels in wireless networks are highly correlated, they require intelligent algorithms to dynamically handle multi-channel access. Reinforcement Learning (RL) algorithms are introduced as effective techniques to optimize network performance. However, current RL methods heavily rely on a computationally intensive deep neural network (DNN) that is not friendly for edge devices. In this paper, we propose HD-RL, a dynamic wireless channel-sharing solution that utilizes brain-inspired lightweight hyperdimensional computing as the learning engine. HD-RL mimics important brain functionalities towards high-efficiency and noise-tolerant computation. HD-RL naturally encodes and memorizes prior knowledge to provide the near-optimal policy for channel throughput and α-fairness in the wireless network. Our evaluation shows that HD-RL achieves maximum throughput and fairness while significantly improving efficiency compared to the state-of-the-art DNN-based RL algorithms. In particular, HD-RL achieves more than 20× speedup for reaching the fairness objective. On average, the speedup of convergence time is more than 10× compared to the baseline. Our results also indicate that HD-RL has substantially higher robustness against possible hardware failure, e.g., up to 40% dimension loss in the model.

INDEX TERMS Brain-inspired computing, hyperdimensional computing, medium access control, reinforcement learning.

I. INTRODUCTION

With recent advances in wireless networks, such as a wide application of the Fifth-generation (5G) wireless, we have seen a clear increasing trend in both network quality and the number of wireless devices. In addition, applications such as self-driving cars and IoT-based city infrastructure significantly increase the density of network devices and the intensity of network activities. According to data from Ericsson and GAO [1], mobile data usage was around 30 billion gigabytes per month in 2019, and the predicted usage for 2025 is over 150 billion gigabytes per month. With this fast-growing trend, the current spectrum resource is still not enough for the increasing spectrum demand. Recently, many prior works have tried to approach the resource allocation problem at different levels of wireless networks, i.e., the physical, network, and medium access control (MAC) layers.

In this paper, we focus on improving the network performance with modifications on the MAC layer using dynamically controlled network nodes, i.e., dynamic MAC protocols. Prior works in this direction mainly utilize machine

The associate editor coordinating the review of this manuscript and approving it for publication was Adnan Kavak.

learning algorithms to achieve model-free control in the network or other system optimization problems [2], [3], [4], [5], [6], [7]. When the dynamics of the system is known, conventional ML algorithms can reach optimal or near-optimal results. For example, Markov chain models [8] can solve a dynamic multi-channel access problem with uncorrelated channels and known transfer probability. Many recent studies took advantage of Reinforcement Learning (RL) algorithms enhanced with Deep Neural Networks (DNN), called Deep RL, to maximize network performance without prior knowledge, e.g., unknown MAC protocols [9], [10], [11].

Although deep RL methods can improve network utilization, they often rely on complex DNN models that lead to inefficient and slow RL training and prediction. This makes deep RL methods impossible for online learning within network nodes, especially when the network dynamics is time-variable. One alternative approach is to offload DNN training to the cloud server and only update the model periodically. However, this inevitably causes delay, and the models in the edge may not have an optimal solution for current network dynamics. In addition, transmitting data to the cloud leads to scalability, security, and privacy concerns.

Instead of using DNN-based RL with heavy computation like backpropagation, we exploit brain-inspired HyperDimensional Computing (HDC) as an alternative paradigm in dynamic wireless channel sharing. HDC mimics important brain functionalities towards highly efficient learning and noise-tolerant computation [12], [13], [14], [15], [16], [17]. HDC is motivated by the observation that the human brain operates on high-dimensional data representations. In HDC, we encode regular input to vectors with high dimensions, i.e., hypervectors. HDC has contributed to multiple lightweight solutions for ML applications, ranging from robotics [18], [19] to biology [20], [21], [22]. A key advantage of HDC is its training capability in one or few shots, where object categories are learned from a few examples as opposed to many iterations. Besides accuracy, HDC is amazingly tolerant of errors, as it operates over random hypervectors, independent and identically distributed. This property is preserved by the encoding operations; hence, a failure in a component is not "contagious". These features make HDC a promising solution for effectively and efficiently learning on edge devices.

In this paper, we present HD-RL, a self-learning MAC protocol with faster learning capability that is suitable for edge devices in the network. We make the following contributions in this paper:

- To the best of our knowledge, HD-RL is the first dynamic wireless channel-sharing algorithm based on hyperdimensional computing. With lightweight HDC operations in hyperdimensional reinforcement learning, our channel-sharing algorithm brings higher efficiency compared to the Deep Q Network (DQN), thereby enabling online learning in an edge environment.
- We apply our HD-RL algorithm to solve the problem of time-domain spectrum sharing when the MAC protocol in a network is unknown. Without prior knowledge, this dynamic protocol will guide the node in choosing the best time to send packets. Thus, it avoids channel collision with limited spectrum resources and achieves maximum channel throughput. By modifying the objective function during the training of the HD-RL model, we can also achieve an optimal α-fairness that prevents one node from fully occupying the channel.
- We compare our HD-RL with the DQN-based algorithm and the theoretically optimized results for spectrum sharing quality and runtime. HD-RL not only achieves near-optimal results but also significantly improves the convergence time of dynamic channel sharing (Section IV-E). Our experiment on Raspberry Pi shows that HD-RL is also more efficient in the edge environment, which is appealing to the practical deployment of this self-learning dynamic MAC protocol.
- We explore the robustness of HDC against hardware failures. We also present the effect of the hypervector dimension on the effectiveness of the HD-RL algorithm. This feature is crucial for wireless network stability since the nodes in the network are in different conditions, and their hardware may not work under the best conditions. By ensuring high robustness, HD-RL MAC protocol will function as expected under interference (Section IV-G).
- Our extensive evaluation shows that HD-RL achieves maximum throughput and fairness while significantly improving the efficiency compared to the state-of-the-art Deep RL algorithms. For example, HD-RL achieves more than 20× speedup for reaching the fairness objective. Our algorithm is also efficient on embedded processors, which achieves more than 10× speedup in terms of convergence on average. In addition, our results indicate that HD-RL has substantially higher robustness against possible hardware failure, e.g., up to 40% bit loss in the model.

The rest of our paper is organized as follows. Section II reviews the related works and provides the background of HDC and RL. Section III describes the proposed HD-RL algorithm. The experimental results are presented in Section IV. Finally, we conclude the paper with Section V.

II. RELATED WORKS AND BACKGROUND

A. HYPERDIMENSIONAL COMPUTING

Brain-inspired HDC is a more energy-efficient method compared to widely utilized neural networks, especially because neural networks are usually deep in layers for most learning tasks. As is well-recognized, DNNs are costly during model training both in terms of runtime and energy costs. On the other hand, in HDC-based learning algorithms, model training or learning cost are largely decreased [14], [23], [24]. There are mainly two characteristics of HDC that lead to efficient learning: (1) mapping the lower dimension input into

the higher dimension space and representing the information holistically; (2) the update of the model is carried out with hardware-friendly and lightweight HDC operations [25]. The latter is the fundamental reason behind the efficiency of HDC-based algorithms, which we will briefly introduce below.

In the hyperdimensional space created along with the HDC encoding process (more details on encoding are provided in Section III-B), there exist a huge number of different, nearly orthogonal hypervectors with dimensionality in the thousands [26]. These hypervectors are holographic and (pseudo)random with i.i.d. components. This lets us combine such hypervectors into a new hypervector using well-defined vector space operations while keeping the information of the two with high probability.

Here we assume $\vec{\mathcal{H}}_1$, $\vec{\mathcal{H}}_2$ are two random hypervectors with dimensionality D. They are generated using the standard normal distribution, that is, $\vec{\mathcal{H}} \in \{h\}^D$ and $h \sim \mathcal{N}(0, 1)$. Therefore, it is easy to observe that the dot product between two hypervectors: $\delta(\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2) \approx 0$. With these assumptions made, we define the following HDC mathematics to model brain functionalities:

- **Binding** (*) of two hypervectors $\vec{\mathcal{H}}_1$ and $\vec{\mathcal{H}}_2$ is done by component-wise multiplication (XOR in binary) and denoted as $\vec{\mathcal{H}}_1 * \vec{\mathcal{H}}_2$. The result of the operation is a new hypervector that is dissimilar to its constituent vectors i.e., $\delta(\vec{\mathcal{H}}_1 * \vec{\mathcal{H}}_2, \vec{\mathcal{H}}_1) \approx 0$; thus, binding is well suited for associating two hypervectors. Binding is used for variable-value association and, more generally, for mapping.
- **Bundling** (+) operation is done via component-wise addition of hypervectors, denoted as $\vec{\mathcal{H}}_1 + \vec{\mathcal{H}}_2$. The bundling is a memorization function that keeps the information of input data into a bundled vector. The bundled hypervectors preserve similarity to its component hypervectors i.e., $\delta(\vec{\mathcal{H}}_1 + \vec{\mathcal{H}}_2, \vec{\mathcal{H}}_1) \gg 0$. Hence, the majority function is well suited for representing sets.
- **Permutation** (ρ) operation, $\rho^n(\vec{\mathcal{H}})$, shuffles components of $\vec{\mathcal{H}}$ with n-bit(s) rotation. The intriguing property of the permutation is that it creates a near-orthogonal and *reversible* hypervector to $\vec{\mathcal{H}}$, i.e., $\delta(\rho^n(\vec{\mathcal{H}}), \vec{\mathcal{H}}) \simeq 0$ when $n \neq 0$ and $\rho^{-n}(\rho^n(\vec{\mathcal{H}})) = \vec{\mathcal{H}}$. Thus, we can use it to represent *sequences* and *orders*.
- **Reasoning** is done by measuring the similarity of hypervectors by calculating the dot product. In this paper, we denote this similarity with $\delta(\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2)$.

Prior works have used HDC mainly to solve classification and cognitive tasks, such as language, text and voice recognition [27], [28], [29], [30], bio-related tasks [20], [22], [24], [31], [32], [33], [34], [35], [36], latent semantic analysis [37], and graph reasoning [25], [38], [39]. For instance, the work in [37] proposes to utilize HDC with random indexing for text classification instead of the unscalable latent semantic analysis. The work in [20] uses HDC-based bio-signal processing to help recognize gestures. In the highlighted machine learning, HDC has achieved higher accuracy with fewer training examples compared to state-of-the-art machine learning solutions, e.g., support vector machines and neural networks [13], [18], [40], [41]. Recently, researchers in this field have also introduced brain-inspired computing to RL algorithms [19], [42], [43], optimizing the cost and quality of learning. However, their focus is on classic control tasks and toy video games, where the whole environment is deterministic and easy to model. Unlike all prior works, this paper is the first effort to focus on the wireless spectrum sharing problem in a highly dynamic environment, and we propose an efficiency-oriented solution utilizing HDC. Our solution achieves better utilization of the spectrum while significantly improving the computational cost and robustness compared to existing deep RL techniques.

B. REINFORCEMENT LEARNING

Reinforcement learning is usually considered the third type of machine learning method, in addition to supervised and unsupervised learning [44]. The reasoning behind this is that training data in RL does not have labels and true values as in the supervised learning case. However, it is also hard to categorize RL as unsupervised learning since it relies on feedback information, such as rewards. RL is also unique because its model learning process is intertwined with model inference; in other words, the learning is through trial-and-error interaction with the environment. The goal of RL is to acquire the best action policy for an environment that is either unknown or known to the agent, which corresponds to either a model-free or model-based setting; RL without environment models is harder to train and more challenging.

Major categories of RL algorithms include value-based and policy-based methods. The value-based method learns a value function that evaluates action choices under current observation of the environment. The optimal policy is a greedy one that selects the action with the highest value assuming the value function is also optimal. There are multiple types of value-based RL algorithms, the most frequently used are tabular Q-learning [45] and its DNN-based variants such as DQN [46] and Double-DQN [47]. They all rely on the Q-value function as a proxy policy, which approximates the future accumulated rewards based on a certain state-action pair. For example, with state observation s and action a, the algorithm learns a $Q(s, a) \simeq \mathbb{E}[R_t|S_t = s, A_t = a]$. However, in policy-based RL, the agent directly learns the optimal policy that guides its selection of actions under different conditions. In this paper, we focus on the model-free value-based RL algorithm to solve the spectrum-sharing problem. In value-based RL, one of the characteristics is that training samples can be off-policy and collected from trajectories of different policies. This leads to better sample efficiency compared to the value-based RL.

RL algorithms, especially those backed by DNN learning, have been successfully applied to various fields, e.g., Atari computer games [46], traffic optimization [48] and robot control [49], [50].

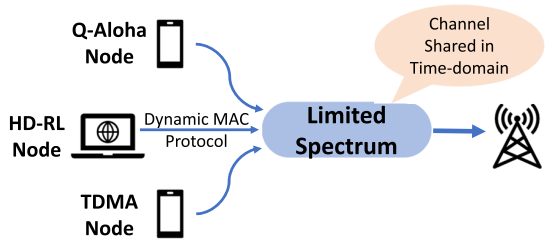


FIGURE 1. Multiple access with HD-RL node and other nodes.

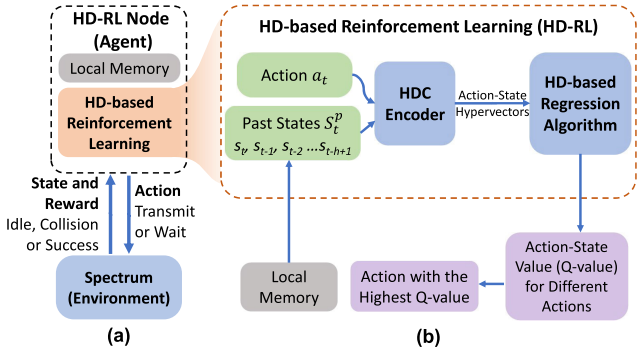


FIGURE 2. Overview of HD-RL framework for spectrum sharing.

C. REINFORCEMENT LEARNING FOR WIRELESS COMMUNICATION AND MAC

Wireless communication is another important area in which RL has been utilized, e.g., for channel selection problems and power consumption optimization [51], [52]. In this paper, we focus mainly on optimizing multiple access decisions to achieve efficient channel usage. Researchers in [9] use DNN-based RL to solve multi-channel access problems with correlated channels and unknown dynamics. With the help of more powerful Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), work in [11] focuses on improving the performance of the heterogeneous wireless network using an RL algorithm. Nodes inside that network have different but also unknown MAC protocols. More recently, researchers have proposed an RL-guided MAC protocol [53] specifically for imperfect channels via a novel feedback recovery mechanism. Another work in [54] aims to solve the hidden terminal problem utilizing Bi-LSTM-powered deep RL. However, the aforementioned works rely on deep RL algorithms, which are computationally expensive. Network dynamics could change fairly fast, and the RL models need regular retraining, so it is not ideal for networks that target real-time and efficient learning. In contrast, we leverage a brain-inspired solution for designing dynamic MAC protocols. Our solution provides fast and energy-efficient computation while also ensuring substantially higher robustness than existing deep RL solutions for spectrum sharing.

III. DYNAMIC MULTIPLE ACCESS USING HD-RL

A. OVERVIEW

Fig. 1 is the overview of the spectrum-sharing problem that we aim to solve using our proposed HD-RL technique. There are various types of spectrum sharing, and we choose to share the limited channel in the time domain, i.e., nodes in the network will be arranged to send packets in different time frames. We use our proposed HD-RL algorithm to achieve a dynamic MAC protocol, and the target is to maximize the utilization of the channel without prior knowledge about other nodes' MAC protocol. Here, we provide more detailed information about the three MAC protocols shown in this figure:

- **TDMA:** Node that applies this protocol only transmits in a few certain slots within each time frame. For example, a TDMA protocol with $X = 5$ guides the node to transmit in the first 5 slots of each 10-slot time frame.
- **q-ALOHA:** A q-ALOHA node has a fixed probability q for transmitting in each time slot.
- **HD-RL:** The HD-RL node is equipped with an HDC-based self-learning agent that determines when the node transmits.

More specifically, our system model is a distributed and heterogeneous network where each node may apply different kinds of MAC protocols. They all transmit packets to an access point and share the wireless channel. Therefore, a node will either be equipped with traditional MAC protocols like q-Aloha and TDMA, or it will be equipped with the HD-RL agent and the proposed dynamic MAC protocol. Nodes with different MAC protocols coexist with each other in the network.

Fig. 2 shows an overview of our proposed HD-RL framework. Fig. 2a has a loop that describes the structure of the HD-RL. The spectrum is the environment that defines states and gives back rewards to the agent for different actions. In our spectrum sharing problem, the spectrum states z are: **Idle**, **Collision**, and **Success**; and the actions a available for agents are: **Transmit** and **Wait**. If two or more nodes in the network choose to transmit packets, we will have a **Collision** in the channel; if all nodes choose to wait, we have the **Idle** state. Both states give back zero rewards because the channel is not well utilized. When there is only one node sending a message, we have the **Success** state and a positive reward with the value 1. Now that we have defined the environment and the action space in the spectrum sharing problem, we next describe the observation/state space of this problem.

Fig. 2b shows the detailed learning process of our proposed HD-RL algorithm. First, it loads a small number of past states $[s_1, s_2, \ldots, s_h]$ from local memory, and we use $h = 10$ in this paper. Within this short history vector, we define each element s_i as the pair of past agent-action a_i and spectrum-state z_i. For example, if the HD-RL node sent a message and observed a collision, we recorded this past state as $s_i = \{a_i, z_i\} = \{\textbf{Transmit}, \textbf{Collision}\}$. Unlike conventional RL tasks that are defined based on the Markov Chain model, i.e., the decision-making process relies solely on the most recent state. On the contrary, we define the state space here as the vector space of the 10 most recent past states, i.e., tuples of agent actions and spectrum states. Notice that the

state s_i defined above does not include any prior knowledge such as the TDMA pattern. Our motivation for using several historical states is that the agent should adjust itself toward an unknown environment. Therefore, we expect the agent to discover patterns of other nodes/users without explicitly acquiring such information. The HD-RL agent listens to the channel status and observes its transmission results to infer patterns of other nodes. The channel status and the HD-RL agent's past actions together can indicate the transmission results of other nodes and channel idleness. Using history states is not unusual; a similar method is also used in solving Atari games using DQN, where people use frame-stacking to combine a few past states so that the agent can capture the necessary information: speed and acceleration of objects.

In the case of spectrum sharing, we consider the inclusion of multiple past states as necessary for the agent to choose the optimal action. For example, if we are going to infer the detailed settings of the TDMA node, which transmits in X out of every 10 slots, a history state of length 10 can effectively capture the pattern. For each element in the history vector, there are 6 different combinations of spectrum states and agent actions. Therefore, the total size of our defined state space is 6^{10} for a history vector of length 10, i.e., about 60 million combinations. This is a particularly large state space and challenging for the traditional tabular-based Q-learning method. In addition, it is also not friendly to the DQN algorithm, because it requires longer training time and leads to larger computation/energy costs.

Given the state history and the next possible action, we develop an *HDC Encoder* that associates and maps states and actions to a single holographic hypervector. Notice that the action a shown in Fig. 3 is the action the agent is planning to take, which is different from the action a_i within the history. We also developed a hyperdimensional regression technique that operates over encoded data and predicts the action-state value, i.e., Q-value, in the case of the DQN method. The action is chosen by selecting the one with the largest action-state value. Next, we will introduce the details of our proposed encoder module and regression algorithm.

In Table 1, we summarize the major mathematical symbols used in this paper.

B. HYPERDIMENSIONAL ENCODING

The encoding is the first step in our HD-RL algorithm. The goal of encoding is to map the original input space to a high-dimensional space. In this section, we propose two encoding methods that are slightly different in how they define the difference between data points.

1) GENERATE STATE AND ACTION HYPERVECTORS

As shown in Fig. 3, we first randomly generate hypervectors for all possible states and actions. $\vec{\mathcal{S}}$ is the hypervector for state s and $\vec{\mathcal{A}}$ is the hypervector for action a. Notice that there are a finite number of states $s = a, z$ because both spectrum states z and agent actions a have a finite number of possibilities. Therefore, we generate random hypervectors uniquely for each possible state s. Thus, for the same two states (s_i and s_j), the encoded hypervector ($\vec{S}_i$ and $\vec{S}_j$) will be the same. As we mentioned in Section II, these hypervectors have a high dimensionality with each element randomly sampled from standard Gaussian distribution, e.g., $\vec{\mathcal{S}}, \vec{\mathcal{A}} \in \{\mathcal{N}(0,1)\}^{\mathcal{D}}$, where $\mathcal{D} \geq 2000$. As a result, the state and action hypervectors are nearly orthogonal, $\delta(\vec{\mathcal{A}}_m, \vec{\mathcal{A}}_n) \simeq 0$ and $\delta(\vec{\mathcal{S}}_m, \vec{\mathcal{S}}_n) \simeq 0$ for $m \neq n$. In this paper, we use real-valued hypervectors (as they are sampled from Gaussian distribution) for greater memorization capacity and to enable better learning quality. This means that the binding operation will be component-wise multiplication instead of XOR operations. Next, we explain the functionality of both encoding modules:

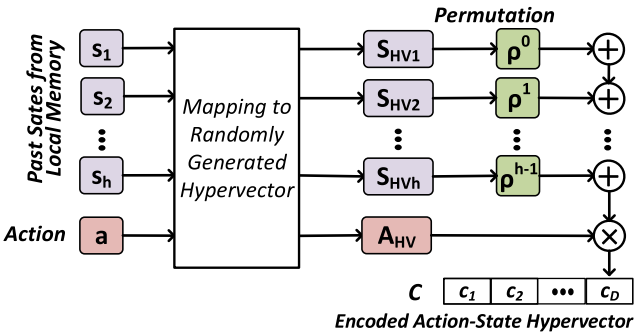


FIGURE 3. Encoding past state history with a selected action.

2) SEQUENCE-BASED ENCODING

It is crucial for networks that use the TDMA protocol to encode different sequences of history state to different hypervectors to extract TDMA patterns. To generate an action-state hypervector, we first bundle different state hypervectors that act as brain-like memorization. We also include permutations with different indices to preserve the temporal location of each state in history. For a given h-step past history S^p, $\{s_1, s_2, \cdots, s_h\}$, we generate high-dimensional state hypervector as:

$$\vec{\mathcal{S}}_\Sigma = \vec{\mathcal{S}}_1 + \rho\vec{\mathcal{S}}_2 + \cdots + \rho^{n-1}\vec{\mathcal{S}}_h \tag{1}$$

where the bundling operand helps to memorize the information of state hypervectors, while the permutation preserves their order in the sequence. For example, the encoded state history of $\{s_1, s_2, s_3\}$ will have a different distribution to the state history of $\{s_2, s_3, s_1\}$:

$$\delta(\vec{\mathcal{S}}_{\Sigma 1}, \vec{\mathcal{S}}_{\Sigma 2}) \simeq 0 \quad \text{where} \quad \begin{cases} \vec{\mathcal{S}}_{\Sigma 1} = \vec{\mathcal{S}}_1 + \rho\vec{\mathcal{S}}_2 + \rho^2\vec{\mathcal{S}}_3 \\ \vec{\mathcal{S}}_{\Sigma 2} = \vec{\mathcal{S}}_2 + \rho\vec{\mathcal{S}}_3 + \rho^2\vec{\mathcal{S}}_1 \end{cases} \tag{2}$$

Unlike other machine learning algorithms that keep state and action separate, our encoding module naturally associates them in high dimension. Particularly, our solution binds the bundled state hypervector with the action hypervector: $\vec{\mathcal{C}} = \vec{\mathcal{S}}_\Sigma * \vec{\mathcal{A}}$. This mapping preserves the information of both state and action while representing it as a distinct and nearly-orthogonal high-dimensional point, i.e., $\delta(\vec{\mathcal{C}}, \vec{\mathcal{A}}) \simeq 0$ and

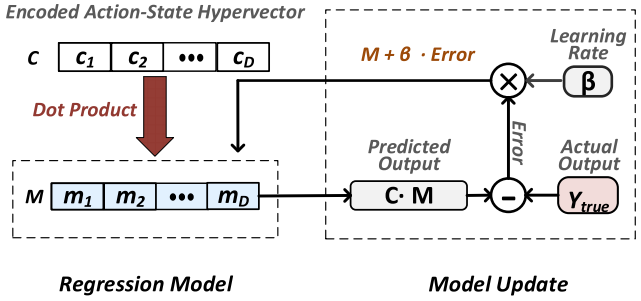


FIGURE 4. Lightweight hyperdimensional regression.

$\delta(\vec{C}, \vec{\mathcal{S}}_\Sigma) \simeq 0$. Since elements in $\vec{S}$ and $\vec{A}$ follow a zero-mean Gaussian distribution, the element-wise multiplication of $\vec{S}$ and $\vec{A}$ will result in a hypervector that gives near-zero dot product with either $\vec{A}$ or $\vec{S}$.

3) ACCUMULATIVE ENCODING

Our second encoding method lies in the fact that temporal or spatial information is not always needed for regression, which is a crucial learning model in our HD-RL. Depending on the task, we might only memorize the accumulative state history without preserving the state sequences. For the spectrum sharing problem, if the node uses the Q-Aloha protocol with unknown probability, we can infer the probability from the percentage of different states within a short history. Thus, we can remove the permutation step and use only the addition of state hypervectors. Given the same h-step history S^p, $\{s_1, s_2, \cdots, s_h\}$, we generate the high-dimensional state hypervector as: $\vec{\mathcal{S}}_\Sigma = \vec{\mathcal{S}}_1 + \vec{\mathcal{S}}_2 + \cdots + \vec{\mathcal{S}}_n$. Similarly to the first encoding, the state and actions can be associated using: $\vec{C} = \vec{\mathcal{S}}_\Sigma * \vec{A}$. The advantage of removing permutation from the encoding is to reduce the runtime on the CPU and prevent memory copy overhead. It also reduces the input space and makes training more stable.

C. HYPERDIMENSIONAL REGRESSION

We develop a hyperdimensional regression model that operates on the encoded action state, $\vec{C} = \{c\}^{\mathcal{D}}$. Fig. 4 shows the overview of our regression, consisting of a single model hypervector. The model hypervector is initialized to all zero elements and has the same dimensionality as the encoded action-state hypervector i.e., $\vec{\mathcal{M}} \in \{0\}^{\mathcal{D}}$. Regression is supervised; thus, we have access to the actual or ground-truth value approximation. In the context of reinforcement learning, the goal of regression is to approximate the ideal Q-function given an encoded action-state hypervector. The ideal Q-function provides a clear expectation of the future cumulative rewards, i.e., the ideal Q-value. For an encoded action-state $\vec{C}$, our model approximates the Q-value by performing the dot product operation between the model and action-state hypervectors: $Y_{pred} = \vec{C} \cdot \vec{\mathcal{M}}$.

During the regression learning process, we expect the predicted value, i.e., the dot product calculated above, to be as close to the ground truth as possible. Therefore, the update of the hyperdimensional regression model should be guided by the error between Y_{pred} and Y_{true} (ground truth). Our model hypervector, thanks to its high dimensionality, can be considered as a memory that returns the predicted value given an input query hypervector. The model training is lightweight because we leverage the element-wise addition to update the hypervector. Our update algorithm for the regression model hypervector is as follows:

$$\vec{\mathcal{M}} = \vec{\mathcal{M}} + \beta \times Error \times \vec{C},$$
$$\text{where } Error = Y_{true} - Y_{pred} = Y_{true} - \vec{C} \cdot \vec{\mathcal{M}} \quad (3)$$

β is a learning rate. This equation ensures the model gets updated with a higher weight for higher prediction error rates ($Y_{true} - Y_{pred} \gg 0$). On the other hand, if the prediction is fairly accurate and the error is small, then the model is only slightly updated with the encoded hypervector. Through iterative training and incremental model updates, our model hypervector memorizes the ground truth value for different inputs and can predict the action-state value accurately. By taking advantage of lightweight HDC operations, such as the dot-product in prediction, our regression accelerates the reinforcement learning process and reaches the optimized spectrum in a fast and efficient way. In section IV-B, we evaluate the efficiency of the HDC-based regression module in HD-RL via a simple regression task.

TABLE 1. Descriptions of mathematical symbols.

Symbol	Description
X	number of transmitting slots in TDMA
q	probability of transmitting in q-ALOHA
a	agent action
z	spectrum-state
s	past state $\{a, z\}$
r	reward
S^p	h-step past history of states
$\vec{\mathcal{S}}$	randomly generated hypervector for s
$\vec{A}$	randomly generated hypervector for a
$\vec{\mathcal{S}}_\Sigma$	encoded hypervector of S^p
$\vec{C}$	encoded action-state pair $\vec{\mathcal{S}}_\Sigma * \vec{A}$
$\vec{\mathcal{M}}_{\text{HDRL}}$	model hypervector of HD-RL
ϵ	probability in ϵ-greedy policy
γ	reward discount
β	learning rate for HD-RL
T	throughput of the network

D. HD-BASED RL FOR DYNAMIC MAC PROTOCOL

In this section, we present our HD-RL technique for the spectrum-sharing problem. As shown in Fig. 2, we use the HDC model to estimate future rewards and then to decide the next action. In reinforcement learning, methods relying on reward prediction are called value-based methods. One of the most commonly used value-based RL is Q-learning, in which a large table or a DNN is used for reward prediction. Our HD-RL follows the main structure of Q-learning.

Due to the limited and discrete action space of this specific task, Q-learning is a great solution with relatively

Algorithm 1 HD-RL for Maximizing Channel Throughput

```
Initialize s_0, ϵ, ϵ_decay, β, γ, τ
Set the past states length h and the training batch size b
Initialize HDC model HDRL, delayed model HDRL'
Initialize local memory MEM
Generate hypervectors for states and actions
for time step t do
    ϵ-greedy algorithm for choosing action a_t:
    if random() ≤ ϵ or len(MEM) < h then
        Randomly choose one action from the action space
    else
        Encode the action-state hypervector C_t
        a_t = argmax_a HDRL(S_t^p, a)
    end if
    ϵ = ϵ × ϵ_decay
    Get reward r_t and new state s_{t+1}
    Record {s_t, a_t, r_t, s_{t+1}} to MEM
    Call function TrainHDRL for HD-RL training
end for
function TrainHDRL(h,b,β,γ,τ)
    Sample an experience batch E of size b from MEM
    for {S_t^p, a_t, r_t, S_{t+1}^p} in E do
        Encode the action-state hypervector C_t
        Calculate predicted q-value Y_t_pred using (4)
        Calculate true q-value Y_t_true using (5)
        Update HDC model HDRL with (6)
    end for
    if step counter reaches τ then
        Reset step counter
        Copy HDRL to delayed model HDRL'
    end if
end function
```

lower computational costs. Actor-critic algorithms, on the other hand, are more suitable for tasks with continuous action space; and the two-part design increases the training/inference cost. The on-policy actor is also slow to learn and not sample-efficient.

For classic protocols, they are unaware of the environment (usage pattern of other nodes) and unable to dynamically adjust themselves to the current supply and demand of spectrum resources. With reinforcement learning, we enable a more efficient access protocol for the agent to behave optimally by inferring the operating protocols of other coexisting networks. Especially, by leveraging HDC-based RL, we can achieve a much faster convergence time for the agent to adjust promptly and in time, leading to a better reduction of collision and utilization of limited resources.

1) MAXIMIZING CHANNEL THROUGHPUT

We first consider one specific objective of optimized spectrum sharing, which is to maximize its throughput. We define throughput as the average number of successfully transmitted data packets per slot. To achieve this objective, we need to reinforce every action that leads to a successful transmission, i.e., every action whose corresponding reward is positive.

In Q-learning methods, we make decisions for the next action by comparing the q-value of each possible action at the current step. In our RL algorithm, we use $\mathcal{HDRL}$ to represent the HDC-based model for action-state value approximation. Thus, we choose the next step/action with $a_t = \text{argmax}_a \mathcal{HDRL}(S_t^p, a)$. The ϵ-greedy algorithm is also included when choosing actions to ensure enough exploration, in which the action is sometimes randomly chosen based on the value of ϵ. As described in Fig. 2, we use multistep past states S_t^p rather than only the most recent state s_t to infer pattern information. With state history length set to h, we have: $S_t^p = \{s_t, s_{t-1}, \ldots s_{t-h+1}\}$. For each action taken, the HD-RL node receives a corresponding reward r_t, depending on whether the packet is successfully sent. Then, the state is updated from s_t to s_{t+1}. We include a FIFO local memory to save this information at each step: $\{s_t, a_t, r_t, s_{t+1}\}$, which is also the source for S_t^p. For simplicity, in the information or experience of step t, we replace s_t and s_{t+1} with S_t^p and S_{t+1}^p even though they span a range of time steps.

To train the HD-RL agent, i.e., update the HD-based regression model, we apply the method called experience replay. Instead of using the most recent S^p only, we randomly sample a batch of S^p at different time steps from local memory. For each past step t and its experience $\{S_t^p, a_t, r_t, S_{t+1}^p\}$, we compare the predicted action-state value from HDC regression and the true value. Experience replay prevents the RL agent from forgetting earlier experiences and balances the distribution of the training sample. Recall Section III-C, $\vec{C}_t$ is the encoded action-state hypervector, and the predicted value is defined as:

$$Y_{t\_pred} = \mathcal{HDRL}(S_t^p, a_t) = \vec{C}_t \cdot \vec{\mathcal{M}}_{\text{HDRL}} \tag{4}$$

For defining the true action-state value in HD-RL, we utilize the Bellman Equation to calculate the temporal difference target, which is widely used in DQN. The Bellman equation decomposes the true value into two parts: the immediate reward r_t and the predicted value for the next step with discount γ. This breaks down the more complex problem and provides an iterative solution.

$$Y_{t\_true} = r_t + \gamma \mathcal{HDRL}'(S_{t+1}^p, \text{argmax}_a \mathcal{HDRL}(S_{t+1}^p, a)) \tag{5}$$

In original Q-learning, the prediction for the next step's value is easily over-estimated and biased, so we also applied a similar solution like Double Q-learning [47]. Notice that we use a different HDC model $\mathcal{HDRL}'$, which is a slow copy of $\mathcal{HDRL}$. We periodically hardcopy the model parameters to update this delayed HDC model. More specifically, after every few steps of model training, we will copy the latest model parameters to the delayed model $\mathcal{HDRL}'$, i.e., its update is delayed and slower than the main model $\mathcal{HDRL}$.

This technique helps us to decouple the selection of the action from its evaluation and mitigate the overestimation of the action values due to the max operation used in the state-action value computation.

After deriving the true and predicted value, we update the regression model hypervector $\vec{\mathcal{M}}_{\text{HDRL}}$ with the prediction error:

$$\vec{\mathcal{M}}_{\text{HDRL}} = \vec{\mathcal{M}}_{\text{HDRL}} + \beta(Y_{t\_true} - Y_{t\_pred}) \times \vec{\mathcal{C}}_t \quad (6)$$

In Algorithm 1, we provide the pseudo-code to maximize channel throughput using HDC. For each time step t, HD-RL makes transmission decisions based on past states and records the reward and the next state in memory. HD-RL training also occurs at each time step, in which the HD-based regression model hypervector $\vec{\mathcal{M}}$ is updated according to the difference between Y_{t_true} and Y_{t_pred}.

2) OPTIMIZING α-FAIRNESS

Unlike the last section in which we directly use reinforcement on positive rewards to achieve maximum throughput, the objective for α-fairness is not equivalent to maximizing rewards. To enable HD-RL for better fairness within the spectrum sharing problem, we embed the α-fairness objective into the update process of a hyperdimensional regression model.

We first define the objective function for α-fairness metric in the network. In this paper, we focus on the condition of $\alpha = 1$ that aims at proportional fairness [55], [56]. The reason for using proportional fairness is that it is a good compromise between max-min fairness and maximum throughput scheduling. For a single node, its α-fairness is given by $\log(T)$ with T referring to the throughput. For a N-node network, the objective is:

$$\text{maximize} \sum_{i=1}^{N} \log(T_i) \quad (7)$$

We now slightly modify the HD-RL training procedure to consider α-fairness. From equation (7) as well as the requirement of achieving fairness, our HD-RL protocol must monitor the throughput of other nodes. The first modification is assigning an HD-based regression model for each node in the network, not only the node equipped with RL-based dynamic MAC protocol. We use these extra models to predict future rewards given to other nodes. The reward serves as a surrogate for the throughput of a node. For example, if there are two nodes in the network (one HD-RL node and one TDMA node), we will assign another $\mathcal{HDRL}_{\text{TDMA}}$ model to predict the Q-value of the TDMA node at each time step. However, this model does not affect the operation of the TDMA node because we do not modify the original TDMA protocol to a dynamic MAC protocol.

The second modification is for equation (5), in which we will embed the fairness objective. Currently, this equation is composed of two parts, the immediate rewards r_t and the maximized future rewards. Notice that this equation

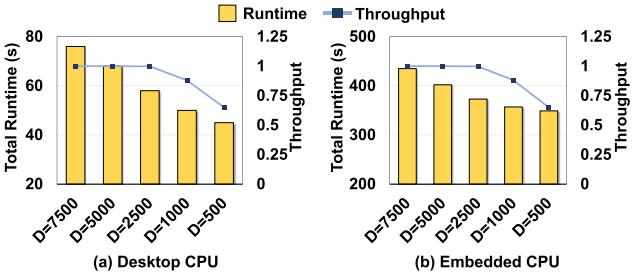


FIGURE 5. Throughput and HD-RL learning runtime with different hypervector dimensionality ranging from 500 to 7500.

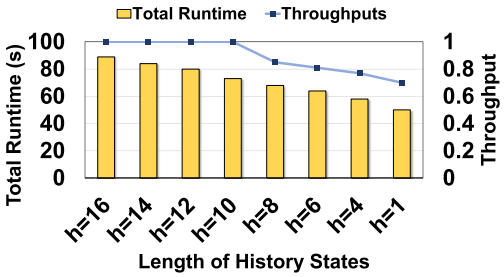


FIGURE 6. HD-RL learning runtime and achieved throughput with different input history length.

involves only a simple accumulation of rewards for the HD-RL node, meaning that its natural goal is to maximize the total throughput. In this case, the HD-RL agent is reinforced to select actions that maximize its transmission success and minimize any collision, thereby leading to maximal throughput. However, the fairness objective requires maximizing the sum of logarithmic rewards from all nodes, which requires more than a simple addition of all rewards. Thus, our modification here is as follows:

$$\text{For Node i: } Y^i_{t\_true} = r^i_t + \gamma \mathcal{HDRL}'^i(S^p_{t+1}, \tilde{a}) \quad (8)$$

To seamlessly embed the fairness objective into the HD-RL training process, we can modify the way we compute the target Q-value, i.e., Y_{t_true}. Previously in equation (5), we chose the action of the next step to maximize the reward of the HD-RL agent only, which failed to consider the α-fairness objective. Therefore, we modify this process to naturally include such objectives. More specifically, to decide the action $\tilde{a}$, we find the sum of logarithmic rewards for each action and choose the one that maximizes the sum:

$$\tilde{a} = \text{argmax}_a\{\sum_{i=1}^{N} \log(\mathcal{HDRL}^i(S^p_{t+1}, a))\} \quad (9)$$

Then, for each node i, the HDC model is updated by adding the encoded input hypervector weighted by prediction error and learning rates:

$$\vec{\mathcal{M}}^i_{\text{HDRL}} = \vec{\mathcal{M}}^i_{\text{HDRL}} + \beta(Y^i_{t\_true} - Y^i_{t\_pred}) \times \vec{\mathcal{C}}_t,$$
$$\text{where} \quad Y^i_{t\_pred} = \mathcal{HDRL}^i(S^p_t, a_t) \quad (10)$$

To conclude, for optimizing α-fairness, we use equation (8) in place of (5) and equation (10) in place of (6). With these modifications, the reinforcement not only aims for larger rewards, but also for a better fairness metric.

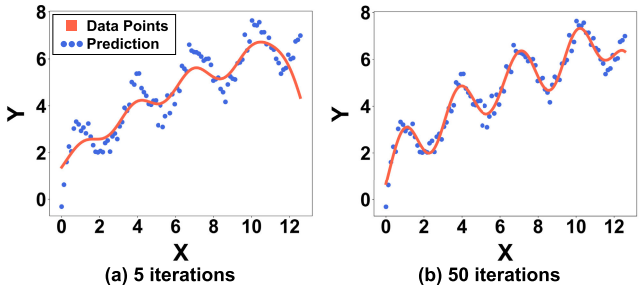


FIGURE 7. Regression quality for different training iterations.

TABLE 2. Regression accuracy and runtime comparison for HDC, neural network, linear regression based algorithms.

Dataset	#Training Epochs	Method	MAE	R2	Train Time (second)	Test Time (second)
Toy Sinusoidal	5	HDC	0.43	0.89	0.005	0.002
		DNN	0.62	0.77	0.77	0.062
	50	HDC	**0.11**	**0.99**	0.039	0.002
		DNN	0.51	0.85	2.78	0.061
	-	LR	0.26	0.96	0.001	0.0004
California Housing	5	HDC	0.5	0.64	0.27	0.003
		DNN	0.54	0.56	42.9	0.11
	50	HDC	**0.45**	**0.68**	2.5	0.003
		DNN	0.51	0.62	437	0.11
	-	LR	0.53	0.58	0.004	0.0008

IV. EXPERIMENTAL RESULT

A. EXPERIMENT SETTINGS

We implement the proposed HD-RL with Python language and verify its functionality. The hardware platforms that support our design in the experiment are AMD Ryzen 5 3600X desktop CPU and Arm embedded CPU in the Raspberry Pi 4. To fully test our HD-RL, we choose several networks with different MAC protocols and parameter settings. We also set up a heterogeneous network with three nodes inside and evaluated the performance of the HD-RL node when coexisting with TDMA and Q-Aloha nodes. For comparison with our design, we choose a DQN-based dynamic MAC protocol [11] that uses ResNet as the Q-learning network. The ResNet in this design has six hidden layers and each layer has 64 neurons. Using the same hardware platform, the DQN results are collected by running the available open-source code. In [57], a benchmark gives the theoretical value of maximum throughput and optimized fairness under different network conditions. This benchmark is model-aware, i.e., it has knowledge about MAC protocol settings such as q and X. We also compare the performance of our design with this benchmark.

Our HDC-based regression model uses hypervectors with $\mathcal{D} = 5000$ dimensions on the desktop CPU and 2500 dimensions on the embedded CPU. As shown in Fig. 5, we explore different hypervector dimensionalities ranging from $\mathcal{D} = 500$ to $\mathcal{D} = 7500$ on both the desktop CPU and the embedded CPU. To illustrate how different sizes of the hypervector influence the learning runtime and quality, we train an HD-RL node that maximizes throughput when another TDMA node is presented. We run the RL process for 5000 steps under different dimensionality settings. We observe that our HD-RL algorithm can achieve maximum 100% throughput when $\mathcal{D} \geq 5000$ while setting $\mathcal{D} \leq 1000$ will significantly lower the throughput due to fluctuation in learning. Compared to $\mathcal{D} = 7500$, lowering the dimensionality to 5k saves nearly 10 seconds in total runtime without any loss of quality, which is why we select $\mathcal{D} = 5000$ for our experiments on the desktop CPU. If we further reduce the dimensionality to 2500, we observe a very small decrease in throughput: on average, less than 0.005. This provides nearly 30 seconds of speedup on the embedded CPU. We believe that this tiny quality loss is acceptable during our experiment on the embedded CPU, where our aim is to minimize runtime.

We use sequence-based encoding for networks with TDMA nodes to ensure our regression model memorizes the sequence of the TDMA pattern, while for Q-Aloha nodes, we choose an accumulative encoding for better efficiency on the CPU. As we mentioned in Section III-A, we set the length of the HD-RL input history states to $h = 10$. Due to our assumption that all TDMA nodes in the system follow a "transmit X out of 10 slots" pattern, we select this particular history length to guarantee the performance of the HD-RL agent. In Fig. 6, we present our quantitative results of the total RL runtime and the throughputs achieved when using different lengths of history states. Here we run RL on a network of one HD-RL node and one TDMA node (X=5: transmits in 5 slots out of 10 with a predefined but unknown pattern). As shown in the figure, h=10 is the minimum required length of history in order to achieve the theoretical maximum throughput of 1. If we use a shorter history, the runtime decreases with the cost of poor learning quality. For example, if we follow the classic Markov decision process model with h=1, then the throughput achieved is just 0.7.

B. EFFICIENCY OF HDC-BASED REGRESSION IN HD-RL

Fig. 7 shows the accuracy of our regression for different training iterations on a simple sinusoidal dataset. We observe that a more significant iteration number provides higher accuracy, i.e., better alignment with the true data. In Table 2, we compare the accuracy and efficiency of the HDC-based, DNN-based, and Ordinary Least Squares based (referred to as Linear Regression—LR in the table) regression algorithms. We evaluate these algorithms on both sinusoidal data and the California Housing dataset [58]. The implementation of DNN and HDC methods are based on optimized numpy code and the LR is based on scikit-learn functions. We set the HDC dimensionality to 500 and use a DNN with two 64-neuron hidden layers with the ReLU activation function. For the sinusoidal dataset, we train algorithms with a training set of 100 data points and test them with 1000 data points. We observe that HDC regression is significantly faster than DNN both in terms of training time and testing time.

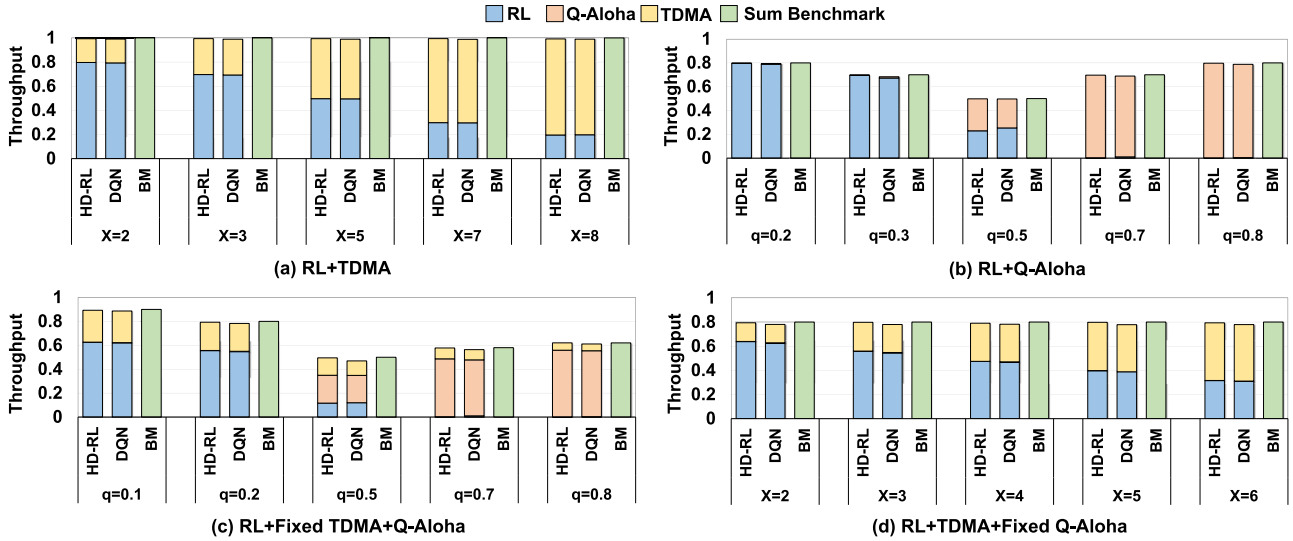


FIGURE 8. Performance comparison between the benchmark and two RL-based MAC protocols for maximizing throughput.

TABLE 3. Maximizing throughput when the RL-based node coexists with a TDMA node.

TDMA Settings	RL Algorithms	RL-based Node Throughput	TDMA Node Throughput	Total Throughput (±2*std)	Theoretical Benchmark
X=2	HD-RL	0.796	0.197	0.993 ± 0.0022	1
	DQN	0.793	0.197	0.99 ± 0.0027	1
X=3	HD-RL	0.696	0.298	0.994 ± 0.0032	1
	DQN	0.693	0.296	0.989 ± 0.0030	1
X=5	HD-RL	0.497	0.497	0.994 ± 0.0016	1
	DQN	0.495	0.494	0.989 ± 0.0012	1
X=7	HD-RL	0.298	0.696	0.994 ± 0.0030	1
	DQN	0.297	0.692	0.989 ± 0.0034	1
X=8	HD-RL	0.196	0.796	0.992 ± 0.0032	1
	DQN	0.198	0.792	0.99 ± 0.0030	1

In addition, it achieves a lower mean absolute error (MAE) and higher R2 value than DNN within 50 epochs. The HDC-based method also converges faster than DNN in terms of epoch number: within 5 epochs, it already achieves lower MAE compared with the DNN results after 50 epochs. As for the LR algorithm, it achieves relatively good accuracy on the toy dataset with the shortest training and testing runtime. However, it is not suitable for regression problems in practice such as predicting the housing price.

C. MAXIMIZING CHANNEL THROUGHPUT

Fig. 8(a) shows the results for the two-node network, in which one node is equipped with the RL-based dynamic MAC protocol and the other one is a TDMA node. We test our proposed HD-RL and the DQN method to maximize the channel throughput, and both of them are compared with the theoretical value from the benchmark (BM in the figure). We also provide the results with different TDMA protocol settings, i.e., changing the number of transmitting slots X within a 10-slot time frame. Notice that the maximum possible throughput for the RL+TDMA condition is always one packet/slot, i.e., the channel is perfectly shared in the time domain. For the HD-RL method, to reach a fully utilized channel, the HDC model has to infer the pattern of the TDMA node with no knowledge about the X value. In Fig. 8(a) and Table 3, we show that our HD-RL learns the TDMA pattern and achieves the theoretical maximum throughput in all five TDMA settings.

Fig. 8(b) and Table 4 show the performance of our HD-RL node when sharing the channel with the Q-Aloha node. We collect the result for multiple protocol settings ranging from $q = 0.2$ to $q = 0.8$. For comparison, we also run the DQN method and the benchmark. When $q < 0.5$, we observe the theoretical throughput when the RL node transmits in every time slot. On the other hand, when $q > 0.5$, the RL node should wait in all slots for maximum overall throughput. For all five settings shown in the figure, our HD-RL is capable of reaching the theoretical maximum.

We also test networks with the three-node setting, i.e., RL node coexisting with Q-Aloha and TDMA nodes. In Fig. 8(c) and Table 5, we fix the TDMA setting ($X = 3$) and explore the Q-Aloha setting from $q = 0.1$ to $q = 0.8$. In Fig. 8(d) and Table 5, we change the TDMA transmitting

TABLE 4. Maximizing throughput when the RL-based node coexists with a Q-Aloha node.

Q-Aloha Settings	RL Algorithms	RL-based Node Throughput	Q-Aloha Node Throughput	Total Throughput (±2*std)	Theoretical Benchmark
q=0.2	HD-RL	0.797	0.001	0.798 ± 0.012	0.8
	DQN	0.79	0.003	0.793 ± 0.010	0.8
q=0.3	HD-RL	0.696	0.002	0.698 ± 0.018	0.7
	DQN	0.673	0.011	0.684 ± 0.016	0.7
q=0.5	HD-RL	0.229	0.269	0.498 ± 0.014	0.5
	DQN	0.253	0.243	0.496 ± 0.014	0.5
q=0.7	HD-RL	0.002	0.695	0.697 ± 0.013	0.7
	DQN	0.01	0.679	0.689 ± 0.016	0.7
q=0.8	HD-RL	0.001	0.796	0.797 ± 0.016	0.8
	DQN	0.004	0.784	0.788 ± 0.017	0.8

TABLE 5. Maximizing throughput when the RL-based node coexists with a Q-Aloha node and a fixed TDMA node.

Q-Aloha Settings	TDMA Settings	RL Algorithms	RL-based Node Throughput	Q-Aloha Node Throughput	TDMA Node Throughput	Total Throughput (±2*std)	Theoretical Benchmark
q=0.1	X=3	HD-RL	0.626	0	0.267	0.893 ± 0.018	0.9
		DQN	0.621	0.001	0.265	0.887 ± 0.016	0.9
q=0.2	X=3	HD-RL	0.555	0.001	0.236	0.792 ± 0.012	0.8
		DQN	0.547	0.003	0.233	0.783 ± 0.014	0.8
q=0.5	X=3	HD-RL	0.117	0.233	0.146	0.496 ± 0.016	0.5
		DQN	0.12	0.229	0.121	0.47 ± 0.014	0.5
q=0.7	X=3	HD-RL	0.002	0.485	0.091	0.578 ± 0.018	0.58
		DQN	0.01	0.468	0.086	0.564 ± 0.018	0.58
q=0.8	X=3	HD-RL	0.001	0.558	0.062	0.621 ± 0.015	0.62
		DQN	0.003	0.551	0.057	0.611 ± 0.016	0.62

TABLE 6. Maximizing throughput when the RL-based node coexists with a TDMA node and a fixed Q-Aloha node.

Q-Aloha Settings	TDMA Settings	RL Algorithms	RL-based Node Throughput	Q-Aloha Node Throughput	TDMA Node Throughput	Total Throughput (±2*std)	Theoretical Benchmark
q=0.2	X=2	HD-RL	0.638	0.001	0.155	0.794 ± 0.014	0.8
		DQN	0.624	0.004	0.152	0.78 ± 0.012	0.8
q=0.2	X=3	HD-RL	0.559	0.001	0.238	0.798 ± 0.010	0.8
		DQN	0.544	0.004	0.232	0.78 ± 0.012	0.8
q=0.2	X=4	HD-RL	0.474	0.001	0.316	0.791 ± 0.008	0.8
		DQN	0.467	0.003	0.311	0.781 ± 0.010	0.8
q=0.2	X=5	HD-RL	0.397	0.001	0.4	0.798 ± 0.012	0.8
		DQN	0.388	0.003	0.39	0.781 ± 0.012	0.8
q=0.2	X=6	HD-RL	0.315	0.001	0.478	0.794 ± 0.013	0.8
		DQN	0.309	0.002	0.468	0.779 ± 0.016	0.8

pattern while keeping the same Q-Aloha setting ($q = 0.2$). We show that our HD-RL achieves near-optimal spectrum sharing for a heterogeneous network with multiple MAC protocols without prior information about the protocols and their settings.

D. α-FAIRNESS OPTIMIZATION

Fig. 9(a)(b) and Table 7 present the result to optimize the spectrum sharing problem with α-fairness objective. We include the test results for both HD-RL and the DQN-based method, which embed the fairness objective into the Q-learning process. The results, as well as the benchmark, are for proportional fairness, i.e., $\alpha = 1$. We test two network conditions for this task: RL co-exists with the Q-Aloha node and RL co-exists with the TDMA node.

As shown in the figure, for both conditions, our HD-RL can optimize the spectrum sharing and achieve near-theoretical α-fairness. For networks where the RL node coexists with the TDMA node, since it is possible to share the spectrum perfectly in the time domain, optimized α-fairness is similar to maximum throughput. As shown in Fig. 9(b), our HD-RL generally provides better fairness than DQN-based RL. In Fig. 9(c) and Table 8, HD-RL follows the proportional fairness objective when $q = 0.2$; and unlike the maximum throughput objective, it ensures the chance for the Q-Aloha node to transmit even with a low q value. When q gradually increases, the throughput of the Q-Aloha node increases

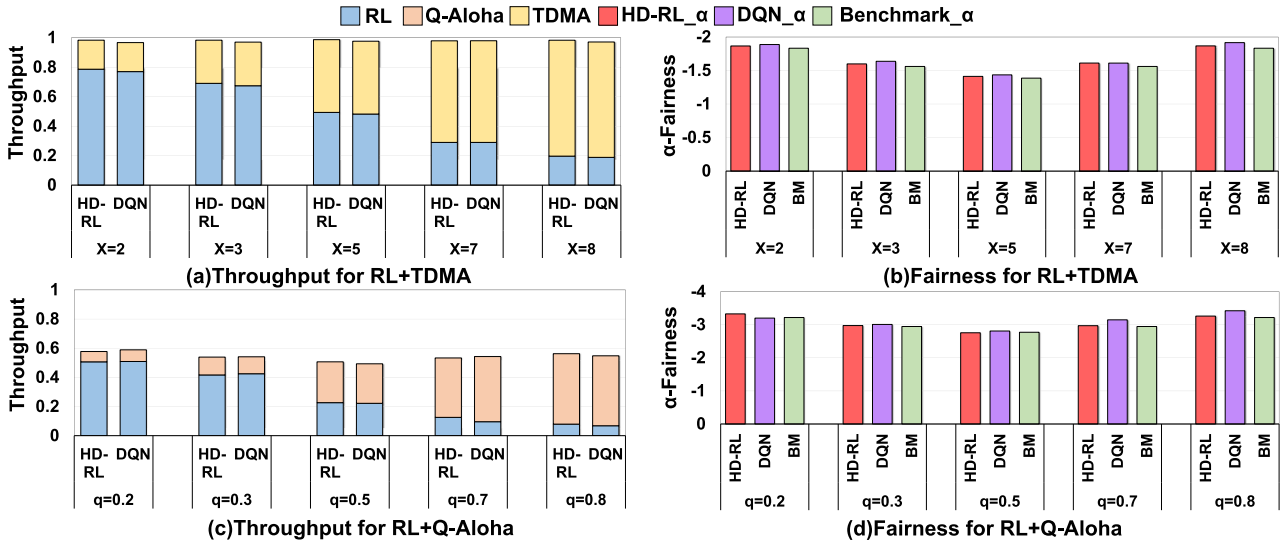


FIGURE 9. Performance comparison between the benchmark and two RL-based MAC protocol for optimizing α-fairness.

TABLE 7. Maximizing fairness when the RL-based node coexists with a TDMA node.

TDMA Settings	RL Algorithms	RL-based Node Throughput	TDMA Node Throughput	Total Throughput (±2*std)	Fairness (±2*std)	Theoretical Benchmark
X=2	HD-RL	0.787	0.197	0.984 ± 0.0038	-1.866 ± 0.0084	-1.833
	DQN	0.77	0.197	0.967 ± 0.0036	-1.888 ± 0.0088	-1.833
X=3	HD-RL	0.69	0.293	0.983 ± 0.0039	-1.598 ± 0.0096	-1.561
	DQN	0.674	0.296	0.97 ± 0.0042	-1.637 ± 0.011	-1.561
X=5	HD-RL	0.493	0.494	0.987 ± 0.0026	-1.413 ± 0.0054	-1.386
	DQN	0.482	0.494	0.976 ± 0.0032	-1.435 ± 0.0064	-1.386
X=7	HD-RL	0.29	0.689	0.979 ± 0.0026	-1.611 ± 0.012	-1.561
	DQN	0.29	0.69	0.98 ± 0.0028	-1.61 ± 0.0084	-1.561
X=8	HD-RL	0.197	0.786	0.983 ± 0.0024	-1.867 ± 0.013	-1.833
	DQN	0.188	0.783	0.971 ± 0.0022	-1.916 ± 0.012	-1.833

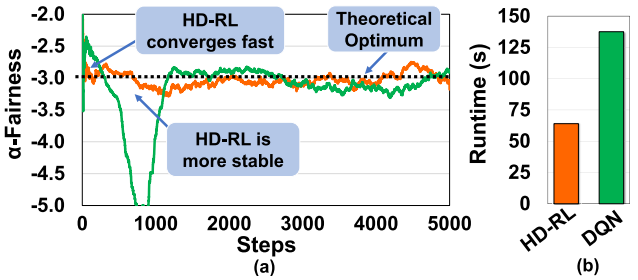


FIGURE 10. Fairness and runtime of network composed of one RL node(HD-RL or DQN) and one Q-Aloha node ($q = 0.3$).

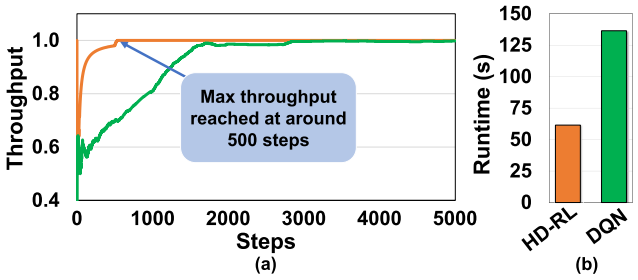


FIGURE 11. Throughput and runtime of network composed of one RL node(HD-RL or DQN) and one TDMA node ($X = 3$).

accordingly, since the RL node transmits less frequently, but the fairness metric prevents the Q-Aloha node from fully occupying the spectrum. We compare our approach with the DQN-based method (Fig. 9(d) and Table 7). Our results indicate that HD-RL provides a better α-fairness for $q \geq 0.3$.

E. EFFICIENCY COMPARISON WITH STATE-OF-THE-ART

Now that we have shown the near-optimized results of our HD-RL in the spectrum-sharing problem, we evaluate the efficiency of our HD-RL and compare it with the DQN-based design. In Fig. 10, we compare the α-fairness achieved by both methods. For overall performance, HD-RL achieves optimized results and is more stable during the first 2000 steps. On the other hand, the DQN-based RL is slow to stabilize and returns significantly lower fairness at around 1000 steps. More importantly, our HD-RL converges to the near-optimal in less than 50 steps, which is more than 20 × faster than the DQN-based method in terms of step number.

TABLE 8. Maximizing fairness when the RL-based node coexists with a Q-Aloha node.

Q-Aloha Settings	RL Algorithms	RL-based Node Throughput	Q-Aloha Node Throughput	Total Throughput (±2*std)	Fairness (±2*std)	Theoretical Benchmark
q=0.2	HD-RL	0.507	0.071	0.578 ± 0.013	-3.328 ± 0.036	-3.219
	DQN	0.509	0.08	0.589 ± 0.013	-3.201 ± 0.035	-3.219
q=0.3	HD-RL	0.417	0.123	0.54 ± 0.011	-2.975 ± 0.058	-2.947
	DQN	0.425	0.116	0.541 ± 0.012	-3.01 ± 0.057	-2.947
q=0.5	HD-RL	0.226	0.28	0.506 ± 0.010	-2.758 ± 0.031	-2.773
	DQN	0.222	0.271	0.493 ± 0.009	-2.811 ± 0.034	-2.773
q=0.7	HD-RL	0.126	0.408	0.534 ± 0.018	-2.972 ± 0.044	-2.947
	DQN	0.096	0.447	0.543 ± 0.017	-3.149 ± 0.046	-2.947
q=0.8	HD-RL	0.079	0.483	0.562 ± 0.011	-3.262 ± 0.073	-3.219
	DQN	0.068	0.48	0.548 ± 0.012	-3.422 ± 0.071	-3.219

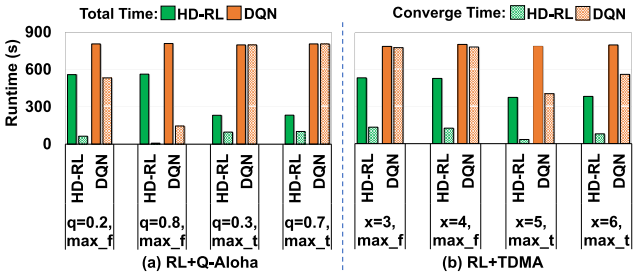


FIGURE 12. Embedded environment runtime comparison between DQN and HD-RL on Raspberry Pi: we test using different settings of Q-Aloha and TDMA and with both optimization target (max_f for maximizing fairness and max_t for maximizing throughput).

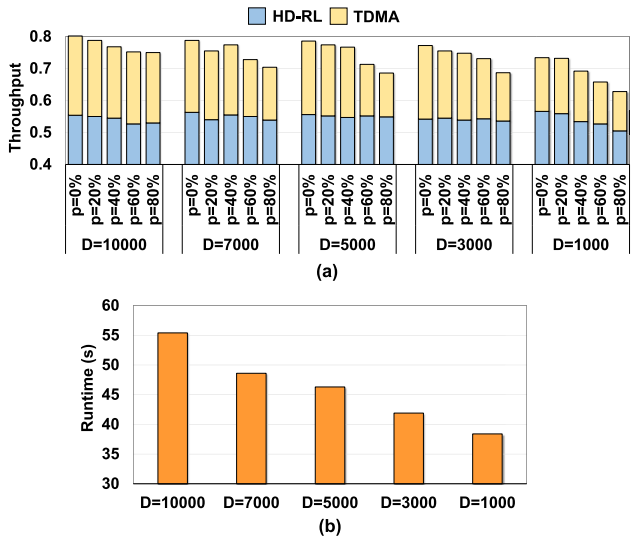


FIGURE 13. Throughput and runtime for different HDC model dimensions and hardware error levels.

In Fig. 10(b), we show that the HD-based method is also 2.1 × faster than the DQN-based method in terms of total runtime for 5000 steps.

As shown in Fig. 11, we also compare two methods to maximize throughput. The network composition is an HD-RL node coexisting with one TDMA node that sends packets in 3 out of 10 time slots. The ideal total throughput is 1, and both methods can achieve it within the test period. However, HD-RL, which represents the red curve, converges to the maximum 3.2 × faster than the other method in the number of steps. As for the total runtime for 5000 steps, we show in Fig. 11(b) that our method is 2.2 × faster.

F. EFFICIENCY OF HD-RL ON AN EMBEDDED PROCESSOR

Apart from evaluating our design on a relatively powerful CPU, we also deploy our HD-RL algorithm on a low-power embedded Arm processor using the Raspberry Pi. As shown in Fig. 12, we run both HD-RL and the baseline DQN algorithms on spectrum-sharing tasks with different network settings. In these tasks, we change the transmitting probability **q** of the Q-Aloha and also the number of transmitting slots **x** in the TDMA. The figure shows that, even in a power and resource-limited environment, HD-RL achieves significantly better efficiency compared to DQN. This indicates that HDC greatly improves the feasibility of RL-based spectrum-sharing algorithms in the edge. We observe that the total HD-RL runtime is up to 3.4× faster than DQN in the Q-Aloha cases. More importantly, when we focus on the time for convergence to theoretical maximum throughput or fairness, HD-RL achieves, on average, 10× faster convergence compared to the DQN baseline. For example, when maximizing throughput in the Q-Aloha case and maximizing fairness in the TDMA case, we notice that it is challenging for the DQN-based algorithm to converge within 5000 steps. However, the HD-RL algorithm achieves maximum fairness and throughput in a very short amount of time.

G. HD-RL ROBUSTNESS

Here, we explore the robustness of HD-RL for the spectrum-sharing problem. Robustness is one of the advantages of computing in high dimensions, and it is also crucial in wireless networks. Meanwhile, HDC algorithms can be deployed and accelerated on emerging processing-in-memory devices to achieve even higher power efficiency. However, these deployment efforts usually suffer from unreliable memory technology that causes stuck-at-zero error or loss of data after extensive memory reads and writes during model training. These errors result in the loss of dimension.

In addition, these errors occur randomly during the operation of the HD-RL algorithm. During HD-RL learning inside network nodes, it is common to have such hardware failures due to a bad environment and radio interference. For our exploration, we assume that both the HDC model and encoded inputs might lose some dimensions/information. Fortunately, in HD-RL, a hypervector contains all the information combined and spread across all its components in a full holistic representation so that no component is more responsible for storing any piece of information than another. We choose the network consisting of one RL node, one Q-Aloha node, and one TDMA node under the settings of $X = 3$ and $q = 0.2$. We ignore the throughput for the Q-Aloha node in the figure, since it is negligible. We run the HD-RL for 3000 steps and under different dimensions and error levels (p refers to the percentage of lost dimensions). Then, we record the average throughput and runtime for each case. In Fig. 13, it is easy to observe that larger dimensions provide better robustness against error. For $\mathcal{D} \geq 5000$, even with 40% dimension loss, the throughput is still close to the optimal value of 0.8. In our test, 20% dimension loss is acceptable for $\mathcal{D} \geq 1000$. We also observe the effect of HDC dimensionality on the quality of the results and the runtime. When the dimension increases from 1000 to 10000, the average throughput also increases. However, it comes at the cost of longer runtime, and thus has a larger energy cost.

V. CONCLUSION

In this paper, we propose HD-RL, the first RL-based MAC algorithm using HDC. We present its application in wireless network spectrum sharing tasks, i.e., an HD-based dynamic MAC protocol optimizes the network for better throughput and fairness. Our HD-RL can achieve near-optimal results in this task and provides significantly better efficiency than DQN-based RL. HD-RL also has the robustness advantage of HDC, which makes it suitable for wireless network applications. As shown in the experiments, HD-RL has proved itself to be a viable and more efficient substitution to the conventional DNN-based RL algorithm. However, the open question is that whether HDC mathematics can be extend to more advanced RL algorithms with entropy regularization and stochastic policy, which theoretically should perform better in the complex dynamics of wireless network systems. The other future direction is to further extend the environment and include more types of MAC protocols, and this helps study the scalability of current discrete action-space RL algorithms.

REFERENCES

[1] U.S. Government Accountability Office. (2020). *5G Wireless: Capabilities and Challenges for an Evolving Network*. [Online]. Available: https://www.gao.gov/products/gao-21-26sp

[2] R. B. Tonetto, H. M. G. D. A. Rocha, G. L. Nazar, and A. C. S. Beck, "A machine learning approach for reliability-aware application mapping for heterogeneous multicores," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[3] A. L. Sartor, P. H. E. Becker, S. Wong, R. Marculescu, and A. C. S. Beck, "Machine learning-based processor adaptability targeting energy, performance, and reliability," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 158–163.

[4] M. Sagi, M. Rapp, H. Khdr, Y. Zhang, N. Fasfous, N. A. Vu Doan, T. Wild, J. Henkel, and A. Herkersdorf, "Long short-term memory neural network-based power forecasting of multi-core processors," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1685–1690.

[5] M. Han, J. Hyun, S. Park, and W. Baek, "Hotness- and lifetime-aware data placement and migration for high-performance deep learning on heterogeneous memory systems," *IEEE Trans. Comput.*, vol. 69, no. 3, pp. 377–391, Mar. 2020.

[6] J. Hyun, J. Park, K. Y. Kim, S. Yu, and W. Baek, "CEML: A coordinated runtime system for efficient machine learning on heterogeneous computing systems," in *Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27–31, 2018, Proceedings 24*. Springer, 2018, pp. 781–795.

[7] N. Valenchon, Y. Bouteiller, H. R. Jourde, X. L'Heureux, M. Sobral, E. B. J. Coffey, and G. Beltrame, "The portiloop: A deep learning-based open science tool for closed-loop brain stimulation," 2021, *arXiv:2107.13473*.

[8] K. Liu and Q. Zhao, "Indexability of restless bandit problems and optimality of whittle index for dynamic multichannel access," *IEEE Trans. Inf. Theory*, vol. 56, no. 11, pp. 5547–5567, Nov. 2010.

[9] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 2, pp. 257–265, Jun. 2018.

[10] S. Xu, P. Liu, R. Wang, and S. S. Panwar, "Realtime scheduling and power allocation using deep neural networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–5.

[11] Y. Yu, T. Wang, and S. C. Liew, "Deep-reinforcement learning multiple access for heterogeneous wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1277–1290, Jun. 2019.

[12] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognit. Comput.*, vol. 1, no. 2, pp. 139–159, Jun. 2009.

[13] Z. Zou, Y. Kim, F. Imani, H. Alimohamadi, R. Cammarota, and M. Imani, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *Proc. Int. Conf. for High Perform. Comput., Netw., Storage Anal.*, Nov. 2021, pp. 1–15.

[14] M. Imani, Z. Zou, S. Bosch, S. A. Rao, S. Salamat, V. Kumar, Y. Kim, and T. Rosing, "Revisiting HyperDimensional learning for FPGA and low-power architectures," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 221–234.

[15] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer, "Computing on functions using randomized vector representations (in brief)," in *Proc. Neuro-Inspired Comput. Elements Conf.*, Mar. 2022, pp. 115–122.

[16] Y. Ni, Y. Kim, T. Rosing, and M. Imani, "Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 292–297.

[17] R. Wang, X. Jiao, and X. S. Hu, "ODHD: One-class brain-inspired hyperdimensional computing for outlier detection," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 43–48.

[18] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Sci. Robot.*, vol. 4, no. 30, May 2019, Art. no. eaaw6736.

[19] Y. Ni, D. Abraham, M. Issa, Y. Kim, P. Mercati, and M. Imani, "Efficient off-policy reinforcement learning via brain-inspired computing," in *Proc. Great Lakes Symp. VLSI*, Jun. 2023, pp. 449–453.

[20] A. Moin, A. Zhou, A. Rahimi, A. Menon, S. Benatti, G. Alexandrov, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt, L. Benini, A. C. Arias, and J. M. Rabaey, "A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition," *Nature Electron.*, vol. 4, no. 1, pp. 54–63, Dec. 2020.

[21] Z. Zou, H. Chen, P. Poduval, Y. Kim, M. Imani, E. Sadredini, R. Cammarota, and M. Imani, "BioHD: An efficient genome sequence search platform using HyperDimensional memorization," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, Jun. 2022, pp. 656–669.

[22] Y. Ni, N. Lesica, F.-G. Zeng, and M. Imani, "Neurally-inspired hyperdimensional classification for efficient and robust biosignal processing," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2022, pp. 1–9.

[23] H. Chen, M. H. Najafi, E. Sadredini, and M. Imani, "Full stack parallel online hyperdimensional regression on FPGA," in *Proc. IEEE 40th Int. Conf. Comput. Design (ICCD)*, Oct. 2022, pp. 517–524.

[24] Y. Ni, Z. Zou, W. Huang, H. Chen, W. Y. Chung, S. Cho, R. Krishnan, P. Mercati, and M. Imani, "HEAL: Brain-inspired hyperdimensional efficient active learning," 2024, *arXiv:2402.11223*.

[25] H. Chen, Y. Ni, W. Huang, and M. Imani, "Scalable and interpretable brain-inspired hyper-dimensional computing intelligence with hardware–software co-design," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, vol. 20, Apr. 2024, pp. 1–8.

[26] S. Aygun, M. S. Moghadam, M. H. Najafi, and M. Imani, "Learning from hypervectors: A survey on hypervector encoding," 2023, *arXiv:2308.00685*.

[27] A. Joshi, J. T. Halseth, and P. Kanerva, "Language geometry using random indexing," in *Quantum Interaction: 10th International Conference, QI 2016, San Francisco, CA, USA, July 20–22, 2016, Revised Selected Papers 10*. Springer, 2016, pp. 265–274.

[28] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "VoiceHD: Hyperdimensional computing for efficient speech recognition," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Nov. 2017, pp. 1–8.

[29] R. Thapa, B. Lamichhane, D. Ma, and X. Jiao, "SpamHD: Memory-efficient text spam detection using brain-inspired hyperdimensional computing," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2021, pp. 84–89.

[30] A. Hernández-Cano, N. Matsumoto, E. Ping, and M. Imani, "OnlineHD: Robust, efficient, and single-pass online learning using hyperdimensional system," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 56–61.

[31] U. Pale, T. Teijeiro, and D. Atienza, "ExG signal feature selection using hyperdimensional computing encoding," in *Proc. IEEE Int. Conf. Bioinf. Biomed. (BIBM)*, Dec. 2022, pp. 1688–1693.

[32] Y. Kim, M. Imani, and T. S. Rosing, "Efficient human activity recognition using hyperdimensional computing," in *Proc. 8th Int. Conf. Internet Things*, Oct. 2018, pp. 1–6.

[33] U. Pale, T. Teijeiro, and D. Atienza, "Systematic assessment of hyperdimensional computing for epileptic seizure detection," in *Proc. 43rd Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Nov. 2021, pp. 6361–6367.

[34] H. Chen and M. Imani, "Density-aware parallel hyperdimensional genome sequence matching," in *Proc. IEEE 30th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2022, pp. 1–4.

[35] H. E. Barkam, S. Yun, P. R. Genssler, Z. Zou, C.-K. Liu, H. Amrouch, and M. Imani, "HDGIM: Hyperdimensional genome sequence matching on unreliable highly scaled FeFET," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Apr. 2023, pp. 1–6.

[36] Y. Ni, Y. Yang, H. Chen, X. Wang, N. Lesica, F.-G. Zeng, and M. Imani, "Hyperdimensional brain-inspired learning for phoneme recognition with large-scale inferior colliculus neural activities," *IEEE Trans. Biomed. Eng.*, early access, Jul. 15, 2024, doi: 10.1109/TBME.2024.3408279.

[37] P. Kanerva, J. Kristoferson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proc. Annu. Meeting Cognit. Sci. Soc.*, 2000, vol. 22, no. 22, p. 22.

[38] P. Poduval, H. Alimohamadi, A. Zakeri, F. Imani, M. H. Najafi, T. Givargis, and M. Imani, "GrapHD: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers Neurosci.*, vol. 16, p. 5, Feb. 2022.

[39] H. Chen, A. Zakeri, F. Wen, H. E. Barkam, and M. Imani, "HyperGRAF: Hyperdimensional graph-based reasoning acceleration on FPGA," in *Proc. 33rd Int. Conf. Field-Program. Log. Appl. (FPL)*, Sep. 2023, pp. 34–41.

[40] G. Karunaratne, M. Schmuck, M. Le Gallo, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, "Robust high-dimensional memory-augmented neural networks," *Nature Commun.*, vol. 12, no. 1, pp. 1–12, Apr. 2021.

[41] M. Nazemi, A. Esmaili, A. Fayyazi, and M. Pedram, "SynergicLearning: Neural network-based feature extraction for highly-accurate hyperdimensional learning," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[42] Y. Ni, M. Issa, D. Abraham, M. Imani, X. Yin, and M. Imani, "HDPG: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, vol. 8, Jul. 2022, pp. 1141–1146.

[43] Y. Ni, W. Y. Chung, S. Cho, Z. Zou, and M. Imani, "Efficient exploration in edge-friendly hyperdimensional reinforcement learning," in *Proc. Great Lakes Symp. VLSI*, Jun. 2024, pp. 111–118.

[44] Y. Li, "Deep reinforcement learning: An overview," 2017, *arXiv:1701.07274*.

[45] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, May 1992.

[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[47] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, vol. 30, no. 1, pp. 2094–2100.

[48] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 3, pp. 1086–1095, Mar. 2020.

[49] Y. Bouteiller, S. Ramstedt, G. Beltrame, C. Pal, and J. Binas, "Reinforcement learning with random delays," in *Proc. Int. Conf. Learn. Represent.*, 2021. [Online]. Available: https://openreview.net/forum?id=QFYnKlBJYR

[50] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 3389–3396.

[51] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Trans. Wireless Commun.*, vol. 18, no. 1, pp. 310–323, Jan. 2019.

[52] Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2239–2250, Oct. 2019.

[53] Y. Yu, S. C. Liew, and T. Wang, "Multi-agent deep reinforcement learning multiple access for heterogeneous wireless networks with imperfect channels," *IEEE Trans. Mobile Comput.*, vol. 21, no. 10, pp. 3718–3730, Oct. 2022.

[54] Y. Shao, Y. Cai, T. Wang, Z. Guo, P. Liu, J. Luo, and D. Gündü, "Learning-based autonomous channel access in the presence of hidden terminals," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 3680–3695, May 2024.

[55] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 556–567, Oct. 2000.

[56] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, Apr. 1998.

[57] Y. Yu, T. Wang, and S. C. Liew. (2019). *Model-Aware Nodes in Heterogeneous Networks: A Supplementary Document to Paper Deep-reinforcement Learning Multiple Access for Heterogeneous Wireless Networks*. [Online]. Available: https://github.com/YidingYu/DLMA/blob/master/DLMA-benchmark.pdf

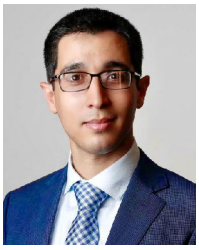
[58] R. K. Pace and R. Barry, "Sparse spatial autoregressions," *Statist. Probab. Lett.*, vol. 33, no. 3, pp. 291–297, May 1997.

YANG NI (Graduate Student Member, IEEE) received the bachelor's degree from the University of Glasgow, and the master's degree from the University of California at San Diego, San Diego, CA, USA. He is currently pursuing the Ph.D. degree with the Department of Computer Science, University of California at Irvine. He is a member with the Bio-Inspired Architecture and Systems Laboratory (BIASLab). His research interests include efficient brain-inspired computing, human-interpretable machine learning, and vector symbolic architecture. He received the Best Paper Award at Design Automation and Test in Europe (DATE) Conference in 2022.

DANNY ABRAHAM received the B.Eng. degree in computer engineering from American University of Beirut, in 2019, and the M.Sc. degree in computer science from the University of California at Irvine, Irvine, in 2022, where he is currently pursuing the Ph.D. degree in computer science, with a focus on the intersection of embedded systems and machine learning.

MARIAM ISSA received the Bachelor of Science degree in math-computer science from the University of California at San Diego, San Diego. She is currently pursuing the Ph.D. degree in computer science. Prior to her graduate studies, she was a Full-Stack Software Engineer of the financial technology space. She is a NSF Graduate Research Fellow with the Donald Bren School of Information and Computer Sciences, University of California at Irvine, Irvine. She is a member with the Bio-Inspired Architecture and Systems Laboratory (BIASLab) and conducts research on hyper-dimensional computing (HDC) and reinforcement learning.

ALEJANDRO HERNÁNDEZ-CANO received the B.Sc. degree in computer science from the National Autonomous University of Mexico, in 2022. He is currently pursuing the M.Sc. degree in computer science with the École Polytechnique Fédérale de Lausanne, Switzerland. His research interests include brain-inspired computing, machine learning, and statistics.



MAHDI IMANI (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in mechanical engineering and electrical engineering from the University of Tehran, and the Ph.D. degree in electrical and computer engineering from Texas A&M University, in 2019. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Northeastern University. His research interests include machine learning, Bayesian statistics, and decision/learning theory, with a wide range of applications from computational biology to cyber-physical systems. He was a recipient of the NIH NIBIB Trailblazer Award in 2022, the Oracle Research Award in 2022, the NSF CISE Career Research Initiation Initiative Award in 2020, and the Best Paper Finalist Award from the 49th Asilomar Conference on Signals, Systems, and Computers, in 2015. He is serving as an Associate Editor for IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.

PIETRO MERCATI (Member, IEEE) received the Ph.D. degree in computer science from UCSD. He is currently a Research Scientist with the Strategic CAD Laboratory (SCL), Intel Labs. He is a SRC Industry Liaison for Intel. His research interests include reliable and energy efficient computing, with an interest in applying machine learning and artificial intelligence to model and optimize computing systems.

MOHSEN IMANI (Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, UCSD. His contribution has led to a new direction on brain-inspired hyper-dimensional computing that enables ultra-efficient and real-time learning and cognitive support. He is currently an Assistant Professor with the Department of Computer Science, UC Irvine, and the Director of the Bio-Inspired Architecture and Systems Laboratory (BIASLab). His research has been recognized with several awards, including the Bernard and Sophia Gordon Engineering Leadership Award, the Outstanding Researcher Award, and the Powell Fellowship Award. He also received the Best Doctorate Research from UCSD and the Best Paper Award in Design Automation and Test in Europe (DATE) and several best paper nomination awards at multiple top conferences, including the Design Automation Conference (DAC) in 2019 and 2020, the Design Automation and Test in Europe (DATE) in 2020, and the International Conference on Computer-Aided Design (ICCAD) in 2020.