CyberRL: Brain-Inspired Reinforcement Learning for Efficient Network Intrusion Detection

Mariam Ali Issa, *Student Member, IEEE*, Hanning Chen, *Student Member, IEEE*, Junyao Wang, *Student Member, IEEE* and Mohsen Imani, *Member, IEEE*

***Abstract*—Due to the rapidly evolving landscape of cybersecurity, the risks in securing cloud networks and devices are attesting to be an increasingly prevalent research challenge. Reinforcement learning is a subfield of machine learning that has demonstrated its ability to detect cyberattacks, as well as its potential to recognize new ones. Many of the popular reinforcement learning algorithms at present rely on deep neural networks, which are computationally very expensive to train. An alternative approach to this class of algorithms is hyperdimensional computing, which is a robust, computationally efficient learning paradigm that is ideal for powering resource-constrained devices. In this paper, we present CyberRL, a hyperdimensional computing algorithm for learning cybersecurity strategies for intrusion detection in an abstract Markov game environment. We demonstrate that CyberRL is advantageous compared to its deep learning equivalent in computational efficiency, reaching up to 1.9× speedup in training time for multiple devices, including low-powered devices. We also present its enhanced learning quality and superior defense and attack security strategies with up to 12.8× improvement. We implement our framework on Xilinx Alveo U50 FPGA and achieve approximately 700× speedup and energy efficiency improvements compared to the CPU execution.**

***Index Terms*—Intrusion Detection, Reinforcement Learning, Hyperdimensional Computing, Brain-inspired Computing, Cybersecurity**

I. Introduction

As we have experienced exponential growth in the technology industry, the prevalence of malicious network-targeted attacks have developed in parallel, making cybersecurity an increasingly critical challenge to address. Prior to the artificial intelligence resurgence, cybersecurity challenges were primarily delegated to domain experts, who were responsible for implementing defense protocols and directly responding to attacks. However, this system quickly grew impractical due to the increased software and infrastructure update frequencies heightening the likelihood of introducing new vulnerabilities.

This work was supported in part by DARPA Young Faculty Award, National Science Foundation #2127780, #2312517, #2321840, #2319198, and #2235472, Semiconductor Research Corporation (SRC), Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research, grants #FA9550-22-1-0253, and generous gifts from Xilinx and Cisco.

Mariam Ali Issa is with the Department of Computer Science, University of California Irvine, Irvine, CA, USA. Email: mariamai@uci.edu

Hanning Chen is with the Department of Computer Science, University of California Irvine, Irvine, CA, USA. Email: hanningc@uci.edu

Junyao Wang is with the Department of Computer Science, University of California Irvine, Irvine, CA, USA. Email: junyaow4@uci.edu

Mohsen Imani is with the Department of Computer Science, University of California Irvine, Irvine, CA, USA. Email: m.imani@uci.edu

A second driving force attributing to this transition is the acceleration in which cyber-attacks are evolving. The advancement in their complexity is why intelligent and automated Intrusion Detection Systems (IDS) and other security defenses have been widely adopted. However, the industry standard is further progressing and moving away from human-in-the-loop to autonomous human-on-the-loop protocols. These solutions are highly sought out due to their ability to adapt to a consistently evolving environment, thus, enabling robust solutions for handling future unforeseen threats [1], [2].

There have been various intelligent solutions for security tasks, ranging from Naive Bayes [3] to neural networks [4], and SVMs [5]; however, these past approaches have required large amounts of high-quality data and careful fine-tuning to achieve adequate detection performance [6]. Furthermore, due to frequent infrastructure changes and the evolution of cyber-attacks, these supervised machine learning based IDS require maintenance for updating the training datasets to handle new attacks.

Differentiating itself from other machine learning methods, Reinforcement Learning (RL) does not require large, curated datasets at initialization. Instead, it learns from interacting with a simulated environment, which is comprised of a set of states, actions, and their respective rewards defined by environment feedback [1], [7]. One popular RL algorithm is Q-Learning, a value-based approach that learns a Q-function for approximating the Q-value (e.g., the expected reward of taking an action at a given state). The original Q-Learning algorithm is Tabular Q-Learning [8], which could only handle simple environments due to the *state explosion problem* [1]. As a result, the neural network based Q-Learning, Deep Q-Network (DQN) [9], was introduced to handle this scalability issue [10].

By implementing the Q-Learning algorithm using a deep neural network, it is able to handle applications with complex learning tasks. However, deep learning algorithms have the following computational challenges: (1) demand unreasonable resources to train since neural networks rely on costly back-propagation and gradient-based methods, (2) are extremely sensitive to noise in data, network, or underlying hardware, and (3) lack human-like cognitive support for long-term memorization and transparency.

To address the aforementioned challenges, recent learning algorithms aim to model the human brain more closely, including HyperDimensional Computing (HDC). Brain-inspired HDC is gaining traction for its impressive learning ability, lightweight hardware implementation, and computational

efficiency [11]–[16]. The origin of this field stems from neuroscience research on how the human brain processes information in high dimension due to the brain's extensive circuitry. HDC abstractly extends this concept as a learning paradigm by modeling and operating on high-dimensional data representations [17].

HDC achieves this by encoding input data into high-dimensional space, outputting what is called *hypervectors*, and using well-defined HDC operations to train and execute inference tasks [11]. Each dimension of the hyperspace abstractly models a neuron's functionality in processing external stimuli [17]. HDC operations are simple arithmetic operations over these dimensions that can be supported on devices with limited resources and computational power [18]. In addition, this class of algorithms is equipped to accomplish both classification and regression machine learning tasks [11], [19]–[21], and has demonstrated comparable results to neural networks while offering higher learning quality (e.g., faster convergence, learning speed, and computational efficiency) [11], [22].

Given these distinctive advantages, HDC suggests itself as a promising potential solution in applications concerning low-powered Internet of Things (IoT) devices [23]–[25]. Most state-of-the-art machine learning approaches involve deep learning, which requires costly computational power to train, thus making them impractical for edge learning [26]. In contrast, HDC-based algorithms are an ideal candidate for their lightweight algorithmic design and energy efficiency [27].

We propose CyberRL, an HDC RL algorithm for developing cybersecurity attack and defense strategies in an abstract intrusion detection Markov game. Our contributions are the following:

- To the best of our knowledge, this paper presents the first HDC based RL algorithm to effectively learn cybersecurity strategies for network intrusion detection.
- We demonstrate CyberRL's ability to learn more effective cybersecurity defense and attack strategies relative to previous Deep Q-Learning algorithms.
- CyberRL is computationally more efficient on multiple computing platforms, including on low-powered edge devices.
- Compared to traditional neural network based RL, CyberRL shows higher hardware efficiency. To further improve CyberRL's performance, we design an FPGA-based domain specific accelerator.

We evaluate the efficacy of our approach on multiple embedded platforms including the NVIDIA Jetson Orin I and the Xilinx Alveo U50 FPGA. Our evaluation shows that CyberRL is computationally more efficient than DQN with a speedup of $1.9\times$ on CPU and $700\times$ on FPGA. In addition, CyberRL is able to achieve $12.8\times$ higher rewards, demonstrating an overall improved learning quality.

II. Background

A. HyperDimensional Computing

HyperDimensional Computing (HDC) is motivated by the theoretical neuroscience observation that the human brain processes information by operating on high-dimensional data representations. This concept formulates the core algorithmic design behind HDC, which is to map objects into their high-dimensional representations.

This is done by encoding the original data into *hypervectors*, which are vectors that store thousands of elements. Each dimension of these hypervectors abstractly constitutes a neuron's functionality in processing external stimuli [17]. Once an HDC model is trained, the encoded hypervector will have captured the important patterns of its respective class across dimensions. The information will be captured throughout the hypervector, which lends the HDC model to also be holographic.

Let us assume that $\vec{\mathcal{H}}_1$ and $\vec{\mathcal{H}}_2$ are two randomly initialized hypervectors, where $\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2 \in \{0,1\}^d$ and d is the dimensionality (e.g., $d = 10,000$). Due to random generation and high-dimensionality, these two vectors are nearly orthogonal: $\delta(\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2) \approx 0$, where δ is a defined similarity function between the two hypervectors.

By working in high-dimensional space, the multitude of nearly orthogonal hypervectors lends itself to be a foundational basis for defining the HDC model. It does so by defining operations, including *bundling* and *binding*, to combine hypervectors while maintaining their respective information with high probability. The training process involves encoding the original input data and *bundling* those of the same class to produce a *class hypervector*, which is the representation of the class in high-dimensional space.

The HDC operation *binding* performs the cognitive computation and learning over the encoded data during the training phase to procure highly accurate, high-dimensional representations. Given a new data point to classify, the inference algorithm is a similarity search conducted between the model hypervectors and the encoded new data point to inform the model in its decision-making. The HDC operations are described in greater detail below:

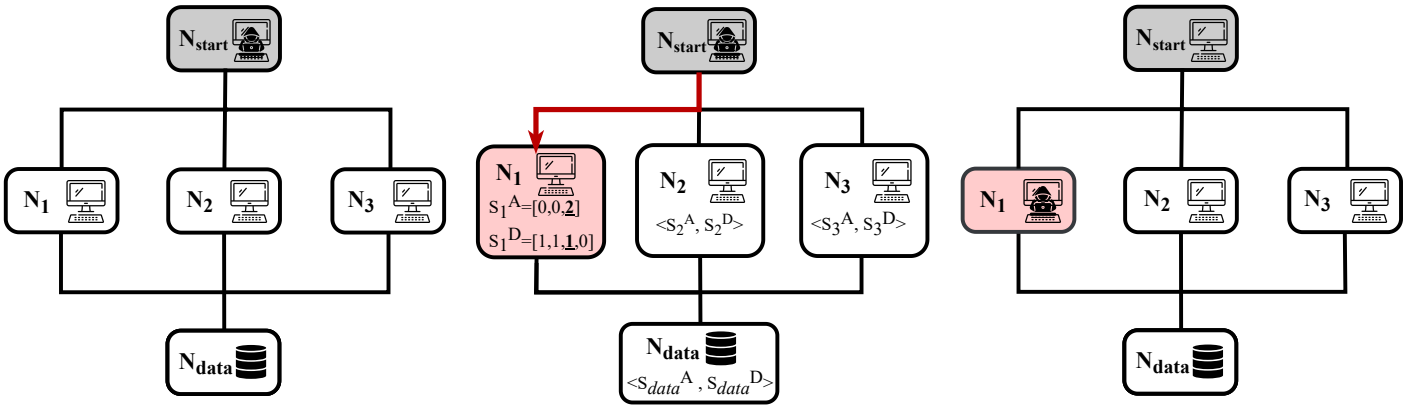
Bundling: The component-wise additive operation of hypervectors, where the information from multiple hypervectors are saved into a single hypervector. This formulates the memorization capability of HDC since the bundled *hypervector* is similar to each of the component hypervectors that comprise it (i.e., $\delta(\vec{\mathcal{H}}_1 + \vec{\mathcal{H}}_2, \vec{\mathcal{H}}_1) \approx 1$).

Binding: The binding operation associates items in hyperspace. Given the hypervectors $\vec{\mathcal{H}}_1$ and $\vec{\mathcal{H}}_2$, component-wise multiplication (XOR in binary) denoted as $(\cdot)$ is conducted to produce a new hypervector that is dissimilar to its constituent vectors (i.e., $\delta(\vec{\mathcal{H}}_1 \cdot \vec{\mathcal{H}}_2, \vec{\mathcal{H}}_1) \approx 0$). This is ideal for constructing hypervectors for variable-value association.

B. Reinforcement Learning

Many Reinforcement Learning (RL) tasks are modeled using the Markov Decision Process (MDP). The MDP is a model of a generally finite system, which can be modified by applied external control. This modeling is discrete, stochastic, and is formally defined as a tuple: $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$, where $\mathcal{S}$ represents the discrete set of states, $\mathcal{A}$ is the discrete set of actions, $\mathcal{T}$ is the transition function, and $\mathcal{R}$ is the reward function [28].

The transition function is defined as $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \to \mathbb{R})$, and the reward function is defined by $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.



(a) Cybersecurity Environment (b) Attacking agent targeting N_1 (c) Attacking agent compromising N_1

Fig. 1. (a) Shows an overview of the topology of a simple computer network to simulate the intrusion detection environment. The attacking agent initially resides at $\mathcal{N}_{start}$ at the beginning of each episode. $\mathcal{N}_{data}$ represents the database containing the network's critical data, which the attacking agent aims to compromise and the defending agent, the Intrusion Detection System (IDS), needs to secure. In this simulation, the network contains a single intermediate layer comprising of three computing devices: $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$ with the connectivity between devices represented by edges. (b) The partially observable state spaces for the agents are shown within each node in the network: $<S_i^A, S_i^D>$, where S_i^A is the attacking agent's cyber-attack repertoire and S_i^D is the IDS's defense to each attack with i indicating the network node. Note that the number of cyber attacks in this elementary simulation is limited to $m = 3$; thus, $|S^D| = 4$ since the defending agent has an additional attribute for the breach detection capacity of the respective node. In (c), the agent is able to successfully compromise $\mathcal{N}_1$ since $S_{1,2}^A = 2 > S_{1,2}^D = 1$. Upon compromising $\mathcal{N}_1$, the attacking agent is now able to target nodes adjacent to $\mathcal{N}_1$ (i.e., $\mathcal{N}_{data}$).

Both of these functions solely depend on the current state and action pair for the transition, ignoring the prior trajectory; this attribute deems the process Markovian [28].

Given an MDP, an environment is defined for an RL task, where the RL agent (i.e., model) learns to analyze the state space to make an informed decision on the next action. The agent relies on the reward probability function $\mathcal{R}$, which measures the agent's success in completing the defined RL task as feedback to improve its overall strategy. There are a number of different approaches in solving this problem statement with the two main classes of RL algorithms being: model-free learning, which comprises value-based RL (e.g., Q-Learning [29]) and policy-based RL (e.g., REINFORCE [30], PPO [31]), and model-based RL (e.g., Dyna-Q [32]).

In regards to this paper, the RL algorithm of greatest interest is Q-Learning. Q-Learning is an off-policy RL algorithm, which learns to approximate the expectation of future rewards (i.e., the *Q-function*) using the Bellman Equation as a condition for optimality [33]. The implementation for CyberRL uses Double Q-Learning, which builds off of the Q-Learning module, and utilizes double estimation to mitigate the overestimation problem. We elaborate further on this algorithm in Section IV.

III. Modeling Intrusion Prevention as a Game

This next section details Hammar and Stadler's abstract simulation of an intrusion detection cybersecurity environment [34]. The task is modeled as a multi-agent zero-sum Markov game, indicating that the positive reward of one agent is strictly at the detriment of the other(s). This particular game environment constitutes two agents: a cyber attacker and an Intrusion Detection System (IDS), the defending agent. The former's objective is to compromise a computer network, while the latter's adversary task is to secure it; in addition to successfully detecting any attempted security breaches by the attacking agent. The game terminates when either agent completes their defined task.

The environment is implemented to simulate a computer network of interconnected computing devices. The node in the first layer, denoted $\mathcal{N}_{start}$, is where the attacking agent resides when the game is initialized. $\mathcal{N}_{data}$, which comprises the last layer, contains the network's sensitive data, which is the node that the defending agent aims to protect from any breaches by the attacking agent. The intermediate layers can include any number of nodes and any defined depth. An illustration of such a structure is presented in Figure 1, where a simplified network is modeled using a single intermediate layer containing three nodes.

The attacker's objective is to reach $\mathcal{N}_{data}$ and successfully compromise it. In order to accomplish this, the attacker must explore the infrastructure through reconnaissance and compromise intermediate nodes until it reaches $\mathcal{N}_{data}$. However, the topology of the infrastructure remains unknown to the attacker with the only accessible information being the successfully compromised nodes and their adjacent neighboring devices. Conversely, the defending agent is aware of the entire network infrastructure, but lacks information pertaining to the status of the attacking agent. Since both agents can only observe a subsection of the entire state space, this formulates a Partially Observable Markov Decision Process (POMDP) [35].

The game is simulated on a round-by-round basis with the defender and attacker alternatively selecting actions to accomplish their respective adversarial objectives. The infrastructure of the game is represented as a graph $\mathcal{G} = <\mathcal{N}, \mathcal{E}>$, where $\mathcal{N}$ denotes the nodes in the network and $\mathcal{E}$ denotes the edges in the graph that represent the devices that are connected to one

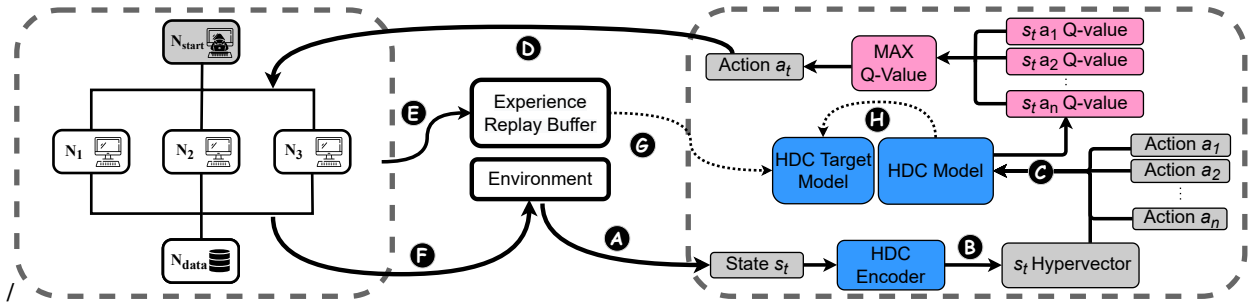


Fig. 2. The above figure outlines the steps of the CyberRL algorithm in the given intrusion detection environment. (A) Given a state s_t, the state is then (B) encoded into a hypervector and (C) multiplied with the class hypervectors, which correspond to each action in the action space (e.g., $a_1, a_2, ..., a_n$). The action corresponding to the maximum Q-value is selected (D) as the next move in the environment. At the conclusion of each step, the given state s_t, the selected action a_t, and the resulting reward r_t, and next state s_{t+1} (E) are stored as a tuple (s_t, a_t, r_t, s_{t+1}) in the experience replay buffer to later be used for (G) updating the CyberRL model. (F) indicates the interaction with the environment, while (H) shows the periodic target model updates.

another. Each node in the network $\mathcal{N}_i \in \mathcal{N}$ has a node state $S_i = <S_i^A, S_i^D>$ to encompass each agent's state space.

Given a node $\mathcal{N}_k$, the attacking agent has a repertoire of cyber-attacks that it can leverage on the targeted node $S_k^A = [S_{k,0}^A, S_{k,1}^A, ..., S_{k,m-1}^A]$, where each element in the set abstractly represents the type of cyber-attack (e.g., Denial-of-Service (DOS) attack, cross-site scripting attack, etc.). Each attribute takes on a numerical value, which represents the strength of the attack and remains unknown to the defending agent.

Similarly, the defending agent has a corresponding set of attributes for node $\mathcal{N}_k$ that map to the defense strength against each cyber-attack: $S_k^D = [S_{k,0}^D, S_{k,1}^D, ..., S_{k,m}^D]$. These represent cybersecurity defenses, such as firewalls or encryption functions. For instance, if attribute j represents a DOS attack, $S_{k,j}^A$ indicates the strength the attacking agent has in leveraging a DOS attack on node $\mathcal{N}_k$. Meanwhile, $S_{k,j}^D$ represents $\mathcal{N}_k$'s resilience from being corrupted by a DOS attack. The final defense attribute in the set for each node $S_{k,m}^D$ is the defending agent's strength in detecting the attacking agent at $\mathcal{N}_k$. The attributes for both the attacking and defending agents are limited to a finite range, specifically $S_{i,j}^A, S_{i,j}^D \in [0, w]$, where w is a natural number.

In terms of the action space, the attacking agent has two courses of action: the first would be to increase the strength of one of its cyber-attacks on any one of the accessible nodes (i.e., increment the value of $S_{k,j}^A$ for node $\mathcal{N}_k$ and attack j) or to attempt to compromise a visible node with an attack of its selection from $0...m-1$. Meanwhile, the defending agent may either choose to strengthen its defense from a particular attack (i.e., increment the value of a defense attribute $S_{k,j}^D$ for node $\mathcal{N}_k$ and counter-attack j) or choose to strengthen the detection ability for any node in the infrastructure (i.e., increment $S_{k,m}^D$ attribute).

If the attacking agent chooses to wage a cyber-attack on node $\mathcal{N}_k$, an attack is simulated by exposing the defending agent's corresponding attribute S_k^D to the attacker. Given that the attacker chose to attack with the attribute j, the attacker would successfully compromise the node if $S_{k,j}^A > S_{k,j}^D$. If the attack is unsuccessful then the defender has an opportunity to identify the attack with a probability of $p = \frac{S_{k,m}^D}{w+1}$, where $S_{k,m}^D$ is the defending agent's detection capability.

For example, in Figure 1 (b-c) the attacking agent is able to successfully compromise $\mathcal{N}_1$ since it leveraged cyber attack $S_{1,2}^A$, which is stronger than $S_{1,2}^D$, the IDS security capability in defending from the respective cyber-attack. If in the scenario that $S_{1,2}^A <= S_{1,2}^D$, then with probability $\frac{S_{1,3}^D}{w+1}$, the IDS would detect the cyber attacker; hence, winning the episode.

The game progresses as each agent takes an action of its choice altering the state environment in each round of the game. The episode terminates when either the attacking agent successfully compromises $\mathcal{N}_{data}$ or when the IDS detects the attacker. The winning agent is awarded +1 utility, while the losing agent receives -1 utility in rewards.

IV. CyberRL as an Intrusion Detection System

In this section, we detail the CyberRL algorithm employed to learn both attack and defense strategies.

As is known with learning new environments in RL tasks, the agent needs to learn undiscovered areas of the state space while also utilizing past experience to learn the best action to take at any given state. This is known as the exploration vs. exploitation trade-off. Exclusively using one of these initiatives results in one of the two extremes: either the model never learns the Q-function or the model converges to a local maxima, resulting in a sub-optimal configuration.

To mitigate this, the epsilon-greedy method is employed, which initializes the RL agent to prioritize exploration at the early stages of training. However, as more experience is accumulated, the agent increasingly relies on the learned Q-function until ϵ completely decays, enabling pure exploitation of its past experience. This is done by initializing $\epsilon = 1$ and applying a decay rate for epsilon to converge to zero. At each time step in training, if the generated number is less than ϵ, then a random action is selected. If not, the model selects the action with the highest Q-value, as shown below:

$$a_t = \begin{cases} \text{random action } a \in \mathcal{A}, & \text{with probability } \epsilon \\ argmax_{a \in \mathcal{A}} Q(s_t, a), & \text{with probability } 1 - \epsilon \end{cases}$$

given an action space $\mathcal{A}$ at time step t.

Similar to Deep Q-Learning, CyberRL also delegates the Q-function approximation to a machine learning model rather

than recordkeeping a Q-table; however, instead of a neural network approximating these values, the agent utilizes an HDC model. This is achieved by mapping the states and actions into hyperdimensional space using the exponential kernel as the encoding function. The HDC model contains n model hypervectors $\{\vec{\mathcal{M}}_{a_1}, \vec{\mathcal{M}}_{a_2}, ...\vec{\mathcal{M}}_{a_n}\}$, where n is the size of the action space (i.e., each of these hypervectors corresponds to an encoded action).

With Deep Q-Learning, an action is selected by choosing the argmax of the highest Q-value computed by the trained neural network. In the HDC version, we do this by multiplying the encoded state hypervector $\vec{\mathcal{S}}_t$ with each model hypervector $\vec{\mathcal{M}}_{a_i}$ to output the corresponding Q-value for each state-action pair. The agent selects the action with the highest Q-value and interacts with the environment, collecting feedback r_t, and observing the next state in this transition s_{t+1}. The algorithm advances to the next time step $t+1$, and iterates until the episode terminates. An overview of this process is detailed in Figure 2.

The class hypervectors of the HDC model are updated over episodes as more experience is accumulated. After each time step t, the state, the selected action, the reward, and the next state are stored in an experience replay buffer as a tuple (s_t, a_t, r_t, s_{t+1}). This experience replay strategy is used to collect additional data for training the agent in an online approach. After sufficient data samples are collected, a batch of these tuples are sampled from the experience replay buffer to update the HDC model $\mathcal{Q}$. Unlike supervised machine learning, which has labeled data at its disposal, the Q-Learning algorithm relies on the Bellman Equation or Dynamic Programming Equation [33], which is a recursive expression for the Q-value at time step t, to estimate the true Q-values. We use this to define the optimal Q-function

$$q_{t\_true} = r_t + \gamma max_a \mathcal{Q}'(s_{t+1}, a) \tag{1}$$

and to maximize the accumulated rewards in an episode.

In order to achieve this, the Bellman Equation states that each sub-task must be optimized to realize the highest rewards for the entire task, which means q_{true} is the sum of r_t and the maximum Q-value for the next time step. As noted in the equation above, we use $\mathcal{Q}'$, the target model, since our approach uses the Double Q-Learning method in updating the model [36]. Compared to $\mathcal{Q}$, which is updated at every time step, $\mathcal{Q}'$ is updated periodically and stabilizes the learning process to avoid overestimating the Q-value, a common consequence of maximizing the Bellman Equation. Additionally, the reward decay γ is included to adjust the weight on future or near-sighted rewards: a value of 1 puts a higher weight for the long-term rewards and a value close to 0 prioritizes the more immediate rewards.

Returning to the update step: for a given tuple (s_t, a_t, r_t, s_{t+1}), we encode the state s_t into hyperdimensional space to give its corresponding hypervector $\vec{\mathcal{S}}_t$ and multiply it with action a_t's class hypervector, $\vec{\mathcal{M}}_{a_t}$, from the HDC model

$$q_{t\_pred} = \mathcal{Q}(s_t, a_t) = \vec{\mathcal{S}}_t \times \vec{\mathcal{M}}_{a_t} \tag{2}$$

We take the difference between q_{pred} and q_{true} and use it to update the HDC model

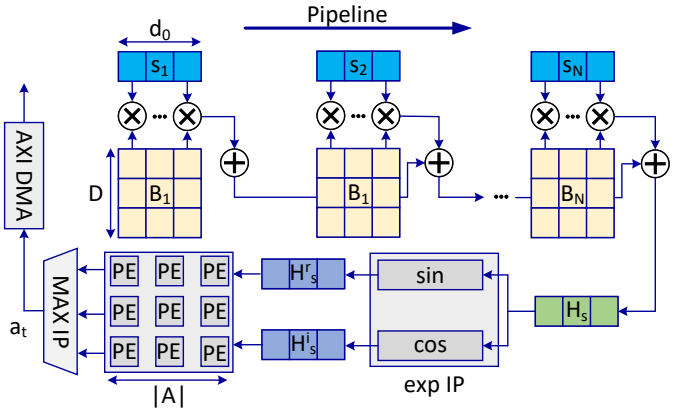


Fig. 3. CyberRL model acceleration on the FPGA. Here s_i represents agent i's original state vector. $\vec{B}$ is the base hypervector matrix. H_s is the encoded state hypervector.

$$\vec{\mathcal{M}}_{a_t} = \vec{\mathcal{M}}_{a_t} + \alpha(q_{t\_true} - q_{t\_pred}) \times \vec{\mathcal{S}}_t \tag{3}$$

where α is the learning rate. Over the duration of learning, the Q-function gradually becomes more accurate in estimating the best actions across the state space.

Furthermore, this development of the HDC Double Q-Learning model includes optimizations in the action selection and model update steps using batch training. This allows for the HDC model to be vectorized in both the Q-value calculations and target model updates. As a consequence, the model experiences significant computational efficiency to enable it to scale to RL tasks with larger state and action spaces.

V. FPGA Architecture Design

Compared to CPU and GPU, HDC models generally have a higher execution performance and energy-efficiency on the FPGA platform [37], [38]. On one hand, FPGA accelerators achieve a balance between computation parallelism and energy efficiency through customized hardware design. On the other, HDC model computation does not require high-precision data computation due to the holographic nature of the HDC model. Essentially, this implies that we can achieve parallel hypervector computation without relying heavily on digital signal processing IP (DSP) and by utilizing lookup tables (LUT) instead.

In Figure 3, we present the FPGA acceleration of CyberRL. Here we assume our model simultaneously controls N agents with each agent's state vector dimensions being $1 \times d_0$. For each agent i, a corresponding base hypervector matrix $\mathbf{B_i}$ with dimensions $d \times D$ is stored in the on-chip storage (such as BRAM). During the state encoding process, we pipeline each agent's state encoding computation. The mathematical representation of this pipeline is:

$$\vec{H}_s = \sum_{i=1}^{N} \vec{d}_i \mathbf{B_i} \tag{4}$$

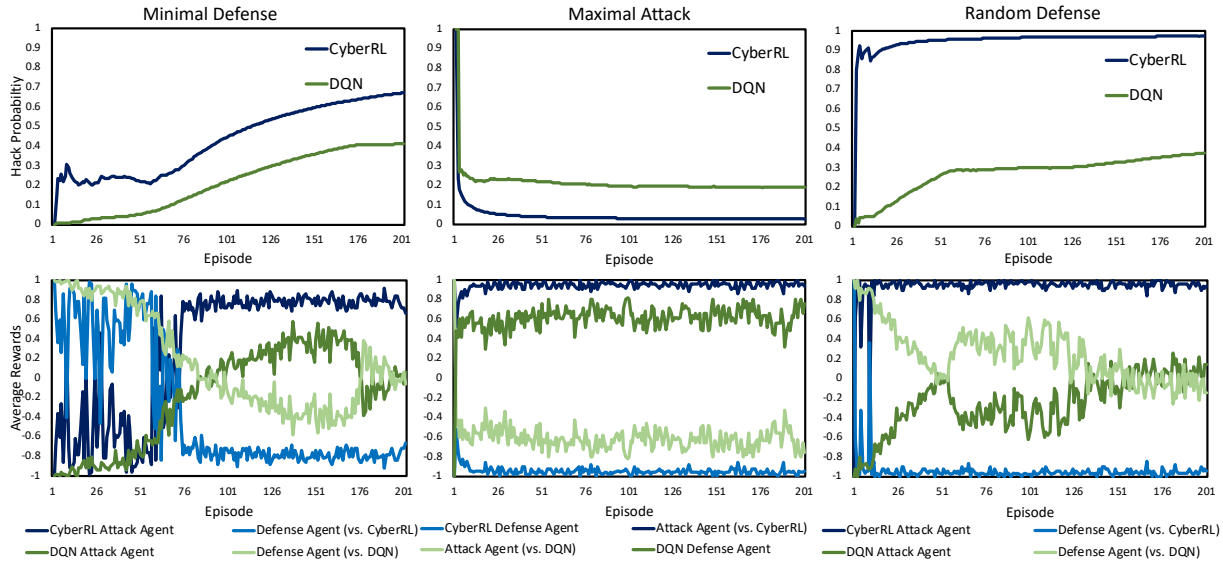


Fig. 4. Comparison of our CyberRL to DQN [9] in the three different intrusion detection scenarios. The top row shows the Hack Probability of the network being compromised, while the bottom row shows the Average Rewards, computed from the last 100 episodes of self-play.

Here the dimension of the encoded state hypervector ($\vec{H_s}$) is $1 \times D$. Before conducting regression operations to select the optimal action, we also pass the encoded state hypervector through a kernel function, such as the exponential function used in CyberRL. To implement the kernel function on-chip, we divide the kernel function IP into two parts, the sine and cosine function components since we have:

$$e^{jx} = cos(x) + jsinx(x) \tag{5}$$

To save on resources, we choose to pre-store sine and cosine values inside the on-chip storage (such as on-chip BRAM) instead of using Xilinx's existing hardware IP, such as CORDIC IP. For the regression part, we accelerate the encoded state and action hypervectors' matrix multiplication using a systolic array [39]. In the last stage, a max IP is used to choose the highest action index. This action index will also be passed back to host CPU via AXI DMA IP.

VI. Experiments

A. Attack and Defense Scenarios

To measure the effectiveness of HDC in learning cybersecurity strategies, we applied our CyberRL algorithm to various intrusion detection scenarios given the simulated environment detailed in Section III, which serves as the dataset [34]. This is a multi-agent RL environment with one agent attempting to compromise the given computer network, while the other acts as an Intrusion Detection System (IDS) (e.g., the attack and defense agents respectively). The given environment includes multiple scenarios to allow for training both of these agents. They are the following:

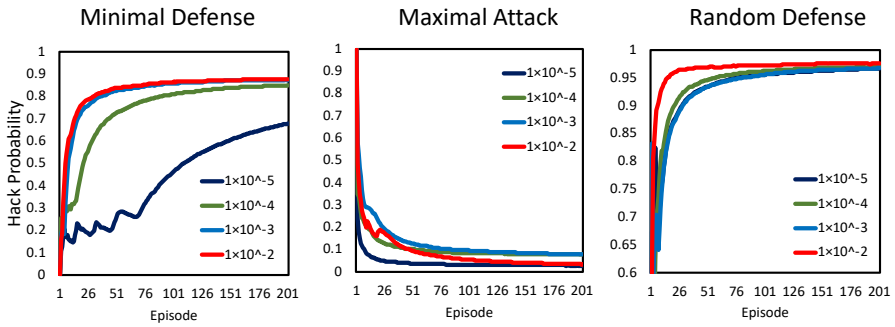
- *Minimal Defense Scenario*: where we train an attacking agent against the environment's IDS, which follows a policy that will always defend the attribute with the minimal value of its neighbors
- *Random Defense Scenario*: where we train an attacking agent against an IDS that uses a random baseline defense policy
- *Maximal Attack Scenario*: where we train the IDS to defend against an attacking agent, who will target the node which has the highest attack attribute

It's important to note that the adversary agent in each of these scenarios will be simulated by the environment. For instance, in the *Maximal Attack* scenario, the adversary agent is the attack agent, which is integrated into the environment, while we train the implemented CyberRL defense agent.

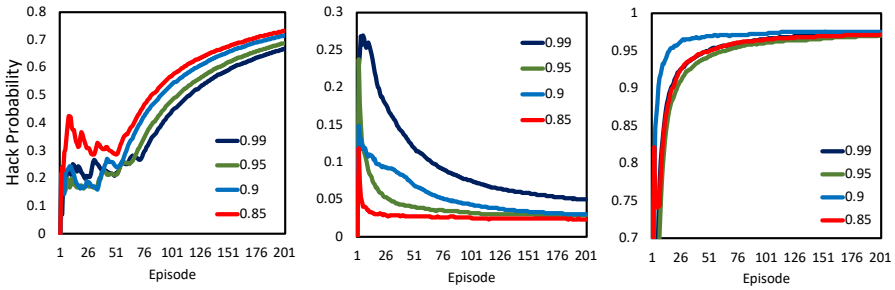
We run our CyberRL algorithm in each of these scenarios and compare it to the Deep Q-Network (DQN) from [9] in terms of computational efficiency and quality of the learned security strategies. The latter is measured by the trained agent's realized Average Rewards during training and Hack Probability.

The Hack Probability metric is the likelihood measure of the computer network environment being hacked. When training a defense agent, as is the case in the *Maximal Attack* scenario, a lower hack probability is desired since it is aligned with the trained model's objective; conversely, for the *Random* and *Minimal Defense* scenarios, a higher hack probability measures a better trained attack agent. The second metric, the Average Rewards, is calculated from the previous 100 game simulations or episodes, where rewards are the measured utility (e.g., -1, +1) of winning the simulated game.

The topology of the network consists of two intermediate layers, each with three nodes, for a total of 8 nodes. Additionally, the attack agent has ten different attack attributes that it can leverage against a node in the network. Hence, the state space of this environment consists of 168 features and the action spaces for the agents are 80 and 88 for the attacking and defending agents respectively. We use the implementation of the DQN from the original project [34] and compare it to



(a) Convergence of different values for α with $\varepsilon, \gamma, Q' update\ frequency$ set to their respective optimal values.



(b) Convergence of different values for ε with $\alpha, \gamma, Q' update\ frequency$ set to their respective optimal values.

Fig. 5. Experiments for fine-tuning the values (a) α and (b) ϵ across the three cybersecurity scenarios.

TABLE I
CYBERRL MODEL HYPERPARAMETERS

	Minimal Defense	**Maximal Attack**	**Random Defense**
HDC Dimension	1000	1000	1000
Learning Rate	0.01	0.00001	0.01
Epsilon Decay	0.85	0.85	0.90
Target Model Update	50	50	50
Discount Factor	0.90	0.95	0.8
Batch Size	32	32	32

TABLE II
NVIDIA JETSON ORIN I PLATFORM. HERE EMC MEANS EXTERNAL MEMORY CONTROLLER. THE VALUE OF EMC REPRESENTS THE EXTERNAL MEMORY USAGE.

Power	**15W**	**30W**	**50W**	**Max**
Frequency	1.1GHz	1.7GHz	1.5GHz	2.8GHz
EMC	5%	6%	6%	2%
Memory	2.2GB	2.4GB	3GB	6.4GB
CPU	4 cores	8 cores	12 cores	12 cores

our CyberRL model, which has been fine-tuned. Table I shows the values for the set hyperaparameters for the final model used to compute the results reported in Figure 4.

B. Experimental Setup

We run our algorithms on a 2.4GHz 8-core Intel Core i9 to report our results. In addition, we deploy our model on the NVIDIA Jetson Orin I with varying power settings. Table II reports the resource utilization of the NVIDIA Jetson Orin I on each of these power settings. As one can observe from Table II, when the power consumption increases, the corresponding frequency, main memory, external memory controller (EMC), and computing cores consequently increase as well. Furthermore, for the FPGA acceleration, we implement the algorithm using C++ HLS and we select Xilinx Alveo U50 as the kernel FPGA board. The communication between host CPU and kernel FPGA is based on the Xilinx Vitis platform [40]. We also use Xilinx Vivado Power Estimator (XPE) to measure the kernel FPGA on-chip power consumption [41].

VII. EVALUATION

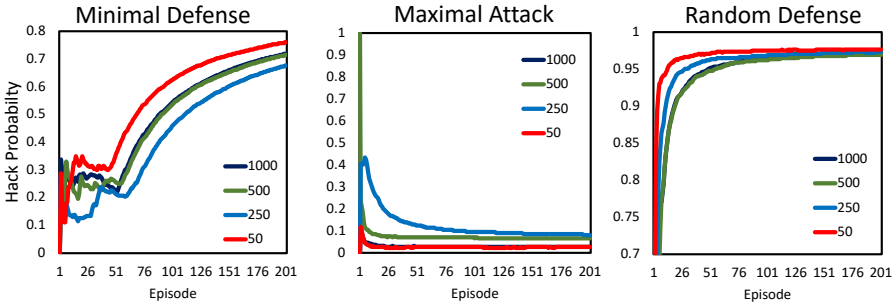
In this section, we demonstrate the efficacy of applying our CyberRL algorithm in finding security strategies in an intrusion detection RL environment. We compare it to DQN and evaluate the performance of the agents in multiple intrusion detection scenarios. We report the computational efficiency and the overhead incurred by the agents' learning process.

A. Hyperparameter Optimization and Training

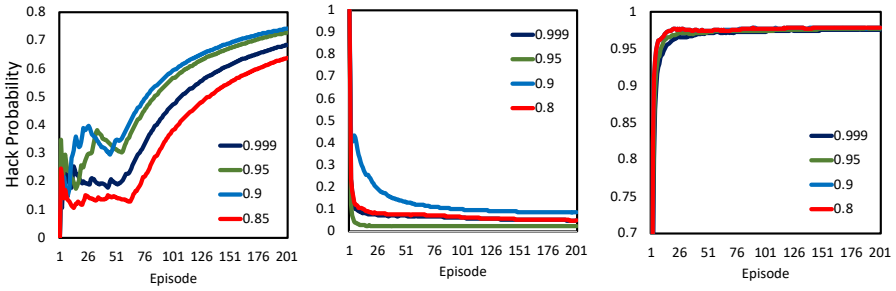
We conducted a thorough exploration of the various hyperparameters for the CyberRL model, which is implemented using a Double Q-Learning framework. The hyperparameters that were fine-tuned include the following:

- α: the learning rate for the HDC model
- ϵ: the epsilon decay for the ϵ-greedy algorithm
- $\mathcal{Q}'$: the target model update frequency for stabilizing the learning curve
- γ: the discount factor for calculating the Temporal Difference Error for the Q-learning algorithm

Figures 5 and 6 display the full set of experiments. The columns correspond to the cybersecurity scenario, while the



(a) Convergence of different values for $\mathcal{Q}'update\ frequency$ with $\alpha, \varepsilon, \gamma$ set to their respective optimal values.



(b) Convergence of different values for γ with $\alpha, \varepsilon, \mathcal{Q}'update\ frequency$ set to their respective optimal values.

Fig. 6. Experiments for fine-tuning the values (a) $\mathcal{Q}'$ and (b) γ across the three security scenarios.

TABLE III
DESIGN ACCELERATION ON ALVEO U50. THE FPGA KERNEL FREQUENCY IS 200MHZ AND FPGA POWER CONSUMPTION IS 17W. HERE $T_{attacker}$ AND $T_{defender}$ ARE SINGLE TIME STEP LATENCY.

Name	LUT	FF	DSP	BRAM	URAM
Total	398.5K	728.8K	7	713	316
Available	872K	1743K	5952	1344	640
Utilization	45.6%	41.8%	∼0%	53.1%	49.3%
$T_{attacker} = 8.94us$, $T_{defender} = 8.64us$					

rows indicate the hyperparameter being optimized. The metric used to determine the optimal value is the *Hack Probability*. The first set of experiments demonstrated in Figure 5 include experiments for determining the values for α and ϵ in each scenario, while Figure 6 includes the experiments for $\mathcal{Q}'$ and γ.

B. Performance

We evaluate the performance between CyberRL and DQN across the three cybersecurity scenarios. Our primary results are presented in Figure 4, which compares the two models over the duration of training. The Hack Probability metrics are presented in the top row of Figure 4 and the Average Rewards for each agent and their adversary agent in the lower row.

Across all three scenarios, the CyberRL model is able to learn a more effective strategy, as shown by achieving a higher Hack Probability when training an attack agent (the left and right columns of Figure 4), and a lower Hack Probability when training a defense agent in *Maximal Attack*. Furthermore, the effectiveness of the CyberRL implementation of the defense and attack agents are reflected in the Average Reward Plots.

The intuition for why CyberRL outperforms the deep learning equivalent stems from HDC's intrinsic memory-like capabilities and pattern recognition from its training framework. By encoding the state and action spaces in hyperdimensional space, CyberRL is advantageous in learning the critical features of the task environment significantly faster than the neural network architecture. These results are not anomalous since HDC-based reinforcement learning algorithms have shown to outperform their deep learning counterparts in various tasks [20], [42], [43].

C. Efficiency

In Table III, we present the resource utilization of our algorithm's implementation on Xilinx Alveo U50 FPGA. For each time step, the attacker agent's execution latency is $8.94\mu s$ and the defender agent's latency is $8.64\mu s$. In addition, Figure 7 plots the CyberRL's improvements in latency and energy efficiency in each of the three scenarios on the CPU and FPGA platforms. The first plot of Figure 7 displays the latency improvements compared to DQN calculated from the training time, which is the duration of time for the model to converge. As anticipated, the CyberRL algorithm is significantly faster in all three scenarios: for the *Maximal Attack Scenario*, the CyberRL is $1.2\times$ more efficient; $1.9\times$ faster in the *Random Defense Scenario*; and $1.6\times$ in the *Minimal Defense Scenario*. When ran on the Xilinx Alveo U50 FPGA, CyberRL achieves around 700× speedup and energy efficiency improvements compared to the CPU execution.

TABLE IV
COMPARISON OF THE COMPUTATIONAL EFFICIENCY BETWEEN THE DQN [9] AND CYBERRL ALGORITHMS ON THE NVIDIA JETSON ORIN I VARIOUS POWER SETTINGS. MEASUREMENTS TAKEN FOR THE MINIMAL DEFENSE SCENARIO AND MEASURED IN SECONDS.

	15W	**30W**	**50W**	**MAX**
DQN	1.9×10^3	1.5×10^3	1.6×10^3	1.0×10^3
CyberRL	1.4×10^3	1.0×10^3	1.3×10^3	0.8×10^3

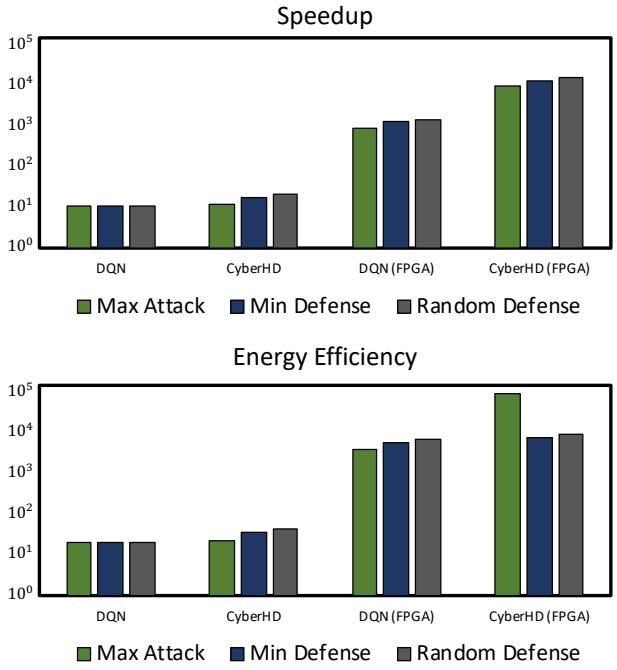


Fig. 7. The above figure displays the time and energy efficiency improvements relative to DQN [9]. (a) shows the CyberRL speedup on both the CPU and FPGA platforms, while (b) shows the improved energy efficiency.

Furthermore, Table IV compares CyberRL and DQN when executed on the NVIDIA Jetson Orin I platform on various low-powered settings (e.g., 15W, 30W, 50W, Maximum Power). We demonstrate CyberRL's efficiency advantage over DQN across all power settings, which also realizes a reduced training time of up to 30%.

VIII. RELATED WORKS

The work from [34] introduces the developed Markov cybersecurity game environment, which demonstrates the efficacy of Deep Reinforcement Learning (DRL) algorithms for solving intrusion detection problems. Another paper from this group, which builds upon their previous work, includes [44], where they formulate the intrusion detection task into an optimal stopping problem. This subsequent work continues to utilize reinforcement learning to estimate optimal defense policies; however, the main contribution is primarily in reframing the cybersecurity environment to solve for optimal stopping.

Other recent work in the area include [45], which introduces a DRL framework for learning defense countermeasures to dynamically evolving environments. They tested various DRL algorithms, including Deep Q-Network (DQN), Advantage Actor Critic (A2C), Asynchronous Actor-Critic, and Proximal Policy Optimization (PPO), to demonstrate the efficacy of DRL to proactively detect various cybersecurity threats. This work introduces a new reinforcement learning environment, which extends the OpenAI Gym library; however, their code has yet to be open-sourced.

In the space of HDC reinforcement learning algorithms, CyberRL is an optimized version of the QHD algorithm introduced in [20]. CyberRL is developed from the Double Q-Learning [36] algorithm and similar to [20], CyberRL implements the Q-function approximation with the HDC framework in lieu of a deep neural network. An issue limiting [20] is its inability to scale to solve RL tasks with large state and action spaces. In this prior work, its primary results stem from OpenAI Gym game environments (e.g., Cartpole, Acrobot, and LunarLanding) [46] where the state and action spaces for each task is relatively small. As discussed in Section IV, the optimization of the algorithm enables CyberRL to scale to learn more complex RL tasks, such as the one provided here [34].

IX. CONCLUSION

In this paper we present CyberRL, an efficient, brain-inspired algorithm for learning cybersecurity strategies in an abstract Markov game environment for solving intrusion detection type security problems. We demonstrate that CyberRL is advantageous in both computational efficiency and in producing stronger defense and attack strategies.

ACKNOWLEDGMENTS

This work was supported in part by the DARPA Young Faculty Award, the National Science Foundation (NSF) under Grants #2127780, #2319198, #2321840, #2312517, and #2235472, the Semiconductor Research Corporation (SRC), the Office of Naval Research through the Young Investigator Program Award, and Grants #N00014-21-1-2225 and #N00014-22-1-2067. Additionally, support was provided by the Air Force Office of Scientific Research under Award #FA9550-22-1-0253, along with generous gifts from Xilinx and Cisco.

REFERENCES

[1] H. Alavizadeh, H. Alavizadeh, and J. Jang-Jaccard, "Deep q-learning based reinforcement learning approach for network intrusion detection," *Computers*, vol. 11, no. 3, p. 41, 2022.
[2] M. Sewak, S. K. Sahay, and H. Rathore, "Deep reinforcement learning for cybersecurity threat detection and protection: A review," in *Secure Knowledge Management In The Artificial Intelligence Era*. Springer International Publishing, 2022, pp. 51–72.
[3] N. B. Amor, S. Benferhat, and Z. Elouedi, "Naive bayes vs decision trees in intrusion detection systems," in *Proceedings of the 2004 ACM symposium on Applied computing*, 2004, pp. 420–424.
[4] Y. Bouzida and F. Cuppens, "Neural networks vs. decision trees for intrusion detection," in *IEEE/IST workshop on monitoring, attack detection and mitigation (MonAM)*, vol. 28. Citeseer, 2006, p. 29.
[5] D. S. Kim and J. S. Park, "Network-based intrusion detection with support vector machines," in *International conference on information networking*. Springer, 2003, pp. 747–756.

[6] K. A. Da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. de Albuquerque, "Internet of things: A survey on machine learning-based intrusion detection approaches," *Computer Networks*, vol. 151, pp. 147–157, 2019.

[7] Y. Badr, "Enabling intrusion detection systems with dueling double deep q-learning," *Digital Transformation and Society*, no. ahead-of-print, 2022.

[8] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

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

[10] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[11] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[12] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 64–69. [Online]. Available: https://doi.org/10.1145/2934583.2934624

[13] 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 *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 656–669.

[14] H. Chen, A. Zakeri, F. Wen, H. E. Barkam, and M. Imani, "Hypergraf: Hyperdimensional graph-based reasoning acceleration on fpga," in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2023, pp. 34–41.

[15] H. Lee, J. Kim, H. Chen, A. Zeira, N. Srinivasa, M. Imani, and Y. Kim, "Comprehensive integration of hyperdimensional computing with deep learning towards neuro-symbolic ai," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

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

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

[18] 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 *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 221–234.

[19] Y. Ni, Y. Kim, T. Rosing, and M. Imani, "Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 292–297.

[20] Y. Ni, D. Abraham, M. Issa, Y. Kim, P. Mercati, and M. Imani, "Efficient off-policy reinforcement learning via brain-inspired computing," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, 2023, pp. 449–453.

[21] Y. Ni, N. Lesica, F.-G. Zeng, and M. Imani, "Neurally-inspired hyperdimensional classification for efficient and robust biosignal processing," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[22] A. Zakeri, Z. Zou, H. Chen, H. Latapie, and M. Imani, "Conjunctive block coding for hyperdimensional graph representation," *Intelligent Systems with Applications*, vol. 22, p. 200353, 2024.

[23] H. Chen and M. Imani, "Density-aware parallel hyperdimensional genome sequence matching," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2022, pp. 1–4.

[24] H. Chen, Y. Kim, E. Sadredini, S. Gupta, H. Latapie, and M. Imani, "Sparsity controllable hyperdimensional computing for genome sequence matching acceleration," in *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2023, pp. 1–6.

[25] E. Hassan, Z. Zou, H. Chen, M. Imani, Y. Zweiri, H. Saleh, and B. Mohammad, "Efficient event-based robotic grasping perception using hyperdimensional computing," *Internet of Things*, vol. 26, p. 101207, 2024.

[26] Y. Lu and L. Da Xu, "Internet of things (iot) cybersecurity research: A review of current research topics," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2103–2115, 2018.

[27] 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 *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[28] W. Uther, *Markov Decision Processes*. Boston, MA: Springer US, 2010, pp. 642–646. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_512

[29] C. Watkins, "Learning form delayed rewards," *Ph. D. thesis, King's College, University of Cambridge*, 1989.

[30] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3–4, p. 229–256, may 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[32] R. S. Sutton, A. G. Barto *et al.*, "Introduction to reinforcement learning," 1998.

[33] R. Bellman, "On the theory of dynamic programming," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 38, no. 8, p. 716, 1952.

[34] K. Hammar and R. Stadler, "Finding effective security strategies through reinforcement learning and Self-Play," in *International Conference on Network and Service Management (CNSM 2020) (CNSM 2020)*, Izmir, Turkey, Nov. 2020.

[35] M. T. J. Spaan, *Partially Observable Markov Decision Processes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 387–414. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3_12

[36] H. Hasselt, "Double q-learning," *Advances in neural information processing systems*, vol. 23, pp. 2613–2621, 2010.

[37] 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 *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 221–234.

[38] Y. Ni, H. Chen, P. Poduval, Z. Zou, P. Mercati, and M. Imani, "Brain-inspired trustworthy hyperdimensional computing with efficient uncertainty quantification," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 01–09.

[39] H. Chen, M. H. Najafi, E. Sadredini, and M. Imani, "Full stack parallel online hyperdimensional regression on fpga," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 2022, pp. 517–524.

[40] V. Kathail, "Xilinx vitis unified software platform," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 173–174.

[41] J. Becker, M. Huebner, and M. Ullmann, "Power estimation and power measurement of xilinx virtex fpgas: trade-offs and limitations," in *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.* IEEE, 2003, pp. 283–288.

[42] M. Issa, S. Shahhosseini, Y. Ni, T. Hu, D. Abraham, A. M. Rahmani, N. Dutt, and M. Imani, "Hyperdimensional hybrid learning on end-edge-cloud networks," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 652–655.

[43] Y. Ni, M. Issa, D. Abraham, M. Imani, X. Yin, and M. Imani, "Hdpg: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2022, p. 1141–1146.

[44] K. Hammar and R. Stadler, "Learning near-optimal intrusion responses against dynamic attackers," 2023.

[45] A. Dutta, S. Chatterjee, A. Bhattacharya, and M. Halappanavar, "Deep reinforcement learning for cyber system defense under dynamic adversarial uncertainties," 2023.

[46] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

X. Biography Section

Mariam Ali Issa (Graduate Student Member, IEEE) is a Computer Science PhD Candidate and NSF Graduate Research Fellow at the University of California at Irvine. She is a member of the Bio-Inspired Architecture and Systems Lab and conducts research on brain-inspired computing paradigms, predominantly HyperDimensional Computing (HDC). Her current research is focused on developing the theoretical foundation of HDC to enable explainability and interpretability.

Prior to her graduate studies, she worked as a full-stack software engineer in the financial technology sector and earned her B.S. Math-Computer Science from the University of California at San Diego.

Hanning Chen (Graduate Student Member, IEEE) is a Computer Science PhD Candidate at the University of California at Irvine. He received the B.S. degree in microelectronic engineering from the Nanjing University, Nanjing, China, in 2019, and the M.S. degree in electrical and computer engineering from Georgia Institute of Technology, Atlanta, GA, USA in 2021. He is a member of the Bio-Inspired Architecture and Systems Lab and conducts research on computer architecture and machine learning.

Junyao Wang (Graduate Student Member, IEEE) received the B.S. degree in mathematics and statistics and the M.S. degree in operations research from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 2019 and 2020, respectively. She is currently pursuing the Ph.D. degree with the Department of Computer Science, University of California at Irvine, Irvine, CA, USA.

Mohsen Imani (Member, IEEE) received the B.Sc. and M.S. degrees from the School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran, in 2011 and 2014, respectively, and the Ph.D. degree from the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, USA, in 2020. He is currently an Assistant Professor with the University of California at Irvine, Irvine, CA, USA. His current research interests include braininspired computing, approximation computing, and processing in-memory.