Beyond von Neumann Era: Brain-inspired Hyperdimensional Computing to the Rescue

Hussam Amrouch[‡], Paul R. Genssler[‡], Mohsen Imani[§], Mariam Issa[§], Xun Jiao[†], Wegdan Mohammad[‡], Gloria Sepanta[‡], and Ruixuan Wang[†]
[‡]University of Stuttgart, Germany, [§]UC Irvine, USA, [†]Villanova University, USA
Corresponding author: amrouch@iti.uni-stuttgart.de

ABSTRACT

Breakthroughs in deep learning (DL) continuously fuel innovations that profoundly improve our daily life. However, DNNs overwhelm conventional computing architectures by their massive data movements between processing and memory units. As a result, novel computer architectures are indispensable to improve or even replace the decades-old von Neumann architecture. Nevertheless, going far beyond the existing von Neumann principles comes with profound reliability challenges for the performed computations. This is due to analog computing together with emerging beyond-CMOS technologies being inherently noisy and inevitably leading to unreliable computing. Hence, novel robust algorithms become a key to go beyond the boundaries of the von Neumann era. Hyperdimensional Computing (HDC) is rapidly emerging as an attractive alternative to traditional DL and ML algorithms. Unlike conventional DL and ML algorithms, HDC is inherently robust against errors along a much more efficient hardware implementation. In addition to these advantages at hardware level, HDC's promise to learn from little data and the underlying algebra enable new possibilities at the application level. In this work, the robustness of HDC algorithms against errors and beyond von Neumann architectures are discussed. Further, the benefits of HDC as a machine learning algorithm are demonstrated with the example of outlier detection and reinforcement learning.

KEYWORDS

Brain-Inspired Computing, Computer Architecture

ACM Reference Format:
Hussam Amrouch, Paul R. Genssler, Mohsen Imani, Mariam Issa, Xun Jiao, Wegdan Mohammad, Gloria Sepanta, Ruixuan Wang. 2023. Beyond von Neumann Era: Brain-inspired Hyperdimensional Computing to the Rescue. In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3566097.3568354

1 INTRODUCTION

The last decade saw a fundamental change in the approach to computing. Instead of compute-heavy tasks, to focus shifted to data-centric applications. Big data, video streaming, and the massive amount of sensor data from IoT devices are only a few examples where data processing is dominant. Machine learning (ML) is another data-centric application, especially during training. Massive datasets are required to achieve the high inference accuracies for large deep neural networks. This fundamental change towards data-centric applications exposes the bottleneck in the predominant von Neumann architecture. The separation of compute and storage creates the *memory wall*, where data transfers dominate the overall power consumption. To address this challenge, custom data-centric architectures have been proposed.

Not only the architecture creates challenges but also transistor technology scaling is reaching its limits. Advanced 2 nm process nodes are impacted by quantum effects, aging effects are magnified, and process variation decreases yield. Emerging technologies such as FeFET are impacted similarly [10]. However, traditional computation methods rely on accurate and error-free hardware. A robust algorithm can relax the demands on the hardware and use it more efficiently. Hyperdimensional computing (HDC) is an emerging ML concept and has proven to be robust against noise and errors from the hardware [29, 35–37]. At the same time, HDC has been explored for in many different application domains, such as circuit reliability [8], modeling of transistor aging [7], natural language processing [27], anomaly detection [30, 31], and bio-informatics [18].

Recently, HDC has been used in anomaly detection, a classical and important application in multiple domains such as IoT, autonomous systems, and finance. It achieves comparable or even better performance than neural networks-based methods such as autoencoder [30, 31]. Another direction of importance in the IoT and edge domain are autonomous systems. Through reinforcement learning, an intelligent agent is trained to act in their environment. Both directions are essential applications for future computing and discussed in this paper. In addition, we focus on hardware implementations of HDC for the beyond von Neumann era.

2 BRAIN-INSPIRED HYPERDIMENSIONAL COMPUTING

2.1 HDC Background

In HDC domain, hyper-vectors (HVs) are the general computing elements which are holographic, high-dimensional (e.g., dimension of HVs $D = 10,000$), and elements in HVs are randomly generated and independent and identically distributed (i.i.d.) [15]. Equation (1) shows a D dimension $\overrightarrow{HV}$ where h_i means the element i in this HV.

$$\overrightarrow{HV} = \langle h_1, \ldots, h_d \rangle \tag{1}$$

In the hyper-dimensional space, for instance the dimension of HV $D = 10000$, two arbitrary HVs are close to orthogonal [15]. The

ASP-DAC 2023, Janurary 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01.
https://doi.org/10.1145/3566097.3568354

purpose of HVs is to integrate and contain information from various scales, and this quasi-orthogonality property of HVs enables HDC to represent and integrate information using different operations.

The two core HV operations in HDC, as Equation (2) illustrated, are bundling and binding operation. The bundling operation and binding operation take two HVs as input and perform element-wise addition and multiplication respectively. The computing output maintains the property of HVs, including high-dimensionality and quasi-orthogonality.

$$\begin{aligned} Bundling(\overrightarrow{HV_i}, \overrightarrow{HV_j}) &= \langle h_{i1} + h_{j1}, \ldots, h_{id} + h_{jd} \rangle \\ Binding(\overrightarrow{HV_i}, \overrightarrow{HV_j}) &= \langle h_{i1} * h_{j1}, \ldots, h_{id} * h_{jd} \rangle \end{aligned} \tag{2}$$

In the meantime, to reveal the relationship between two HVs, HDC utilizes the distance metric δ to measure the similarity between two HVs. As Equation (3) shows, here we use the Hamming distance metric which has the following properties, if the Hamming distance between two HVs is close to 0, the information encoded in the two HVs are highly correlated. On the other hand, if the Hamming distance is close to 1, which means the two HVs are literally not similar. Moreover, the Hamming distance of any two arbitrary HVs is close to 0.5 in the hyper-dimensional space.

$$\delta(\overrightarrow{HV_i}, \overrightarrow{HV_j}) = Hamming(\overrightarrow{HV_i}, \overrightarrow{HV_j}) \tag{3}$$

2.2 Establishing an HDC Model

Three main processes are required to establish an HDC model, named the encoding process, the training process, and the inference process. The following description in this section contains the specific implementation of the each HDC process.

Firstly, encoding is the process to represent a data sample in the hyper-dimensional space with an HV. Encoding is the critical process which may influence the performance of HDC, and therefore variant and sophisticated encoding methods are proposed [6]. Here we use the Record-based encoding strategy as an example. The encoding process requires some extra HVs to encode feature value and feature position in a sample, named item memory (IM) in HDC. The general HDC encoding process can be represented in Equation (4).

$$\overrightarrow{HV_I} = Encoding(I, IM) \tag{4}$$

During the Record-based encoding process, assume as input we have a gray-scale image $I = \langle p_1, \ldots, p_n \rangle$ with n pixels. HDC then randomly generates n quasi-orthogonal base HVs $HV_B = \{\overrightarrow{HV_{B_1}}, \ldots, \overrightarrow{HV_{B_n}}\}$ for each pixel and 256 quasi-orthogonal level HVs $HV_L = \{\overrightarrow{HV_{L_0}}, \ldots, \overrightarrow{HV_{L_{255}}}\}$ representing the value of each pixel in the range of $[0, 255]$. In this case IM contains two sets of HVs IM = $\{HV_B, HV_L\}$. Then, for the encoding of each pixel, HDC binds the corresponding base HV with the level HV. As an example, if the first pixel have value 255, then the corresponding HV (pixel HV) can be computed by $\overrightarrow{HV_{p_1}} = Binding(\overrightarrow{HV_{B_1}}, \overrightarrow{HV_{L_{255}}})$. At the end of encoding process, HDC encode the entire gray-scale image in to a single image HV $\overrightarrow{HV_I} = Bundling(\overrightarrow{HV_{p_1}}, \ldots, \overrightarrow{HV_{p_n}})$ by aggregating all the pixel HVs.

Then, the training process is to build an HDC model over all the training samples, a.k.a associate memory (AM). Considering an m-class classification task, HDC first encodes each training sample into HVs using IM. Then HDC bundles all the training HVs $\overrightarrow{HV^i}$ into m class HVs according to their label i, as illustrated in Equation (5). After the training process, HDC model use m class HVs $\{\overrightarrow{HV_{C_1}}, \ldots, \overrightarrow{HV_{C_m}}\}$ in AM to represent all the training samples.

$$AM = \{\overrightarrow{HV_{C_1}}, \ldots, \overrightarrow{HV_{C_m}}\} = \{Bundling(\overrightarrow{HV^1}), \ldots, Bundling(\overrightarrow{HV^m})\} \tag{5}$$

Finally, the inference process is to determine the category of a testing sample. HDC first encodes the query sample Q into a query HV $\overrightarrow{HV_Q}$ following the same encoding strategy as Equation (4). Then HDC measures the Hamming distance between the query HV $\overrightarrow{HV_Q}$ and the m class HVs $\overrightarrow{HV_C}$ in AM to obtain the inference result. As Equation (6) shows, index of the class HV with the lowest Hamming distance indicates the prediction result y_{pred}.

$$y_{pred} = argmin(\delta(\overrightarrow{HV_Q}, \overrightarrow{HV_{C_1}}), \ldots, \delta(\overrightarrow{HV_Q}, \overrightarrow{HV_{C_m}})) \tag{6}$$

3 FROM ACCELERATION TO IN-MEMORY COMPUTING

HDC is based on large vectors with thousands of bits or numbers. Many hypervectors are required for encoding and classification. The amount of data of all those hypervectors quickly overwhelms smaller caches, causing repeated data transfers. Each transfer reduces the system's throughput, increases power consumption and thus heat [11]. In this section, we describe the potentials for HDC acceleration starting at a special instruction for inference to a dedicated in-memory computing system. Further, we explore the potential for approximate computing and the impact of errors on different memories used for HDC.

3.1 Special Inference Instruction

Each class is represented by one hypervector. During inference for binary hypervectors, the Hamming distance with the query hypervector is computed. This computation comprises two phases, first an XOR operation followed by counting the resulting ones (popcount). While advanced CPU cores have a popcount instruction, low-end CPUs typically lack it. Hence, an accelerator fusing the XOR and popcount operation reduces the latency of an inference.

To evaluate this gain, a custom accelerator is integrated into a RISC-V single-core Rocket CPU. Based on Chipyard [3], this accelerator is exposed as a special instruction to the CPU. The C-code is adopted to use it during an inference. The accelerator is tightly integrated and sends memory requests to the L1 cache. Thus, the data of the hypervectors has to be transferred from the off-chip memory if it is not already present in the caches. Once the (partial) class and query hypervectors are loaded into the accelerator, the XOR and popcount operations are executed and the result stored in a register of the accelerator. If the hypervectors are larger than the internal memory of the accelerator, then they are divided into chunks and the operation is repeated for each chunk.

The Rocket CPU with an L1 cache of 1 KiB and L2 cache of 256 KiB serves as a baseline. The CPU does not have an accelerator nor a popcount instruction. The following three configurations are shown in Fig. 1. A design with very large caches (16 KiB L1, 1024 KiB L2) represents a non-HDC specific approach to increase performance. The accelerator is coupled once with the same-sized

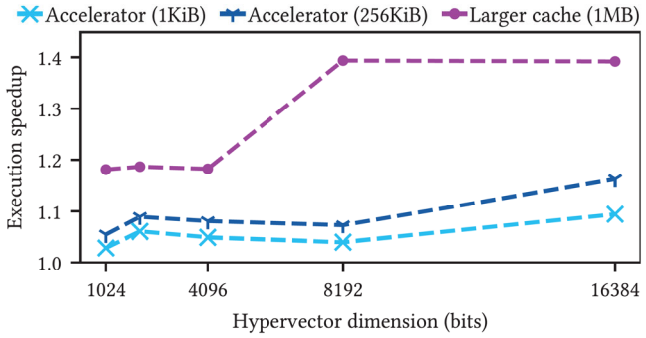

Figure 1: Speedup with a dedicated Hamming distance accelerator or larger caches compared to the baseline.

caches (1 KiB L1, 256 KiB L2) and also with enlarged caches (256 KiB L1, 256 KiB L2). Even though the accelerator speeds up the execution by approximately 5 % to 10 %, very large caches yield even higher speedups of up to 40 %. However, a large SRAM-based on-chip memory comes at a high cost of chip area, almost 2 x and 3.2 x for the enlarged and largest caches, respectively. In conclusion, a dedicated instruction to accelerate the HDC inference computation does not improve the performance as much as larger on-chip memories showing that HDC is memory limited. The resulting large memory footprint of HDC motivates further research in the area of in-memory computing.

3.2 In-memory Design

The previous evaluation focuses on the Hamming distance computation between a query and a class hypervector. However, encoding the real-world data into a query hypervector consumes most of the computation time [9]. Many hypervectors are required during the encoding process, for image processing more than 1000. Therefore, the encoding process has the highest memory footprint and thus generates the most data transfers between the processing unit and the off-chip memory. In this section, we describe an FPGA-based accelerator for encoding and inference, which will also serve as a test bed to evaluate approximate computing and error injection [25]. Other dedicated in-memory computing architectures have been proposed and employ CMOS [28] or emerging technologies [29].

A general overview of the design is shown in Fig. 2. The target platform is an AMD-Xilinx Zynq FPGA XC7Z020 featuring an ARM CPU (called PS) and the FPGA fabric (PL). The external DDR3 memory is accessed through up to four 64-bit high-performance (HP) ports and provides either already encoded query hypervectors or the raw real-world data to be encoded. The experiments were conducted with the image recognition dataset MNIST [17]. It contains 28x28 pixel grayscale images of the ten digits (i.e., ten classes) written by hand. The dataset contains 60 000 images for training and another 10 000 images as test samples.

Similar to the accelerator, a Hamming distance computation core is created for a dimension of 8192-bit hypervectors. If both, class and query, have to be transferred to the FPGA to compute their HIamming distance, then one pass over the test dataset takes 77 ms. The effective memory bandwidth is halved to transfer the query and the class at the same time. More importantly, the class hypervectors

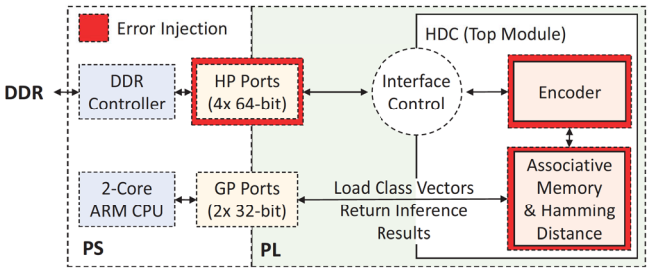

Figure 2: Overview of a Zynq-based FPGA design with optional error injection.

have to be repeatedly transferred every time since they are not stored. To eliminate this overhead, the BlockRAM of the FPGA is utilized to store the ten class hypervectors as an AM. Now only the query has to be transferred and the ten Hamming distances can be computed in parallel. A speedup of more than 10x is achieved by moving the computation and memory closer together.

As described above, the encoding process consumes most of the compute time during an inference. Hence, the FPGA design is extended by an encoder block, including its IM. The 28x28 pixel images in the MNIST dataset can be represented as a features vector of size 784 requiring 784 item hypervectors. Due to this significantly increased demand for on-chip storage, the resources of the FPGA limit the achievable dimension to 1024 bits. Nevertheless, it still enables research for approximate computing and on the impact of errors on HDC.

3.3 Approximate Computing for HDC

Approximate computing aims at reducing the complexity of computations and through that simplifying their implementation in hardware. A large potential has the bundling operation defined through the majority operation. For each component, the number of 1s and 0s in the input hypervectors is counted. Because of the limited width of the memory interface, only a few features are processed per clock cycle. Hence, the intermediate count for each component has to be stored. With a dimension of 1024 bits and 784 features in the MNIST dataset, 1024 10-bit counters are required. One approach to reduce the hardware cost is approximate adders. The results demonstrate a significant loss in inference accuracy with no reduction in resource utilization. The root cause is the mismatch of the bit-width in the operands. The small numbers added per cycle are ignored by many approximate adders. Further research in this direction is required.

Alternatively, instead of approximating the adder, the size of the accumulator to count the 1s can be limited. For most elements in the query vector, the number of 0s and 1s is balanced and does not reach 784. Counting only 1s, the threshold for the majority is 392. If zeros are treated as -1, then the threshold drops to 0. Assuming a balanced stream of 0s and 1s, the accumulator count alternates around 0 and does not require 10 bits. Hence, the bit-width of the accumulator can be reduced to reduce the resource utilization. If the count is larger than the size of the accumulator allows, then it overflows and results in an incorrect bit in the query hypervector. Since HDC is robust against errors, some are tolerable and do not

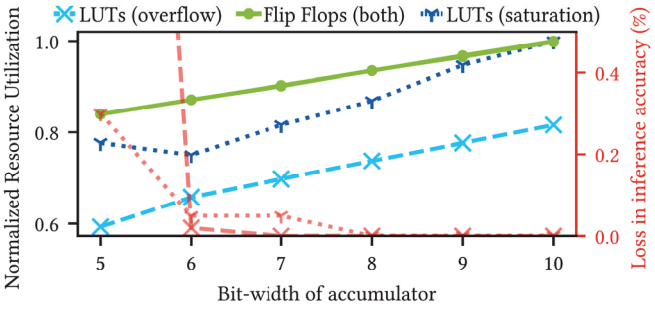

Figure 3: Reduced resource utilization from approximate computing, with an about 3 % loss in inference accuracy with a 5-bit accumulator without overflow checks (dashed). The error can be limited if the accumulator saturates instead (dotted). However, the additionally required LUTs can be saved by using a 6-bit accumulator.

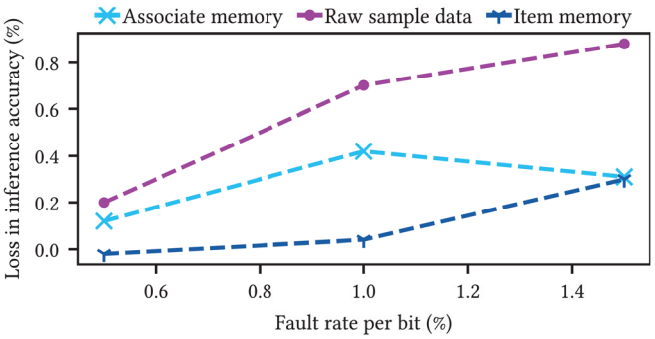

Figure 4: Loss in inference accuracy if the stored item and class hypervectors are subject to bit flip errors. Bit flips in the raw image data have a higher impact.

impact the inference accuracy. To avoid over/underflows, a check is added to limit the count to its maximum or minimum ("saturation" in Fig. 3). Such a check reduces the loss in inference accuracy for 5-bit accumulators from 3.3 % to 0.3 %. However, additional resources (LUTs) are required. Depending on the overall utilization of the FPGA, 6-bit accumulators offer a tradeoff. The increase in flip flips can be compensated by a decrease in LUTs, because the saturation checks are no longer necessary.

3.4 Error injection

HDC is known to be very robust against noise and errors from the underlying memory [16] or computations [29]. Several sources of errors have been explored, each with different errors models, probabilities, and location of the errors. Because such models are probabilistic, an evaluation has to be repeated many times to get statistically meaningful results. Further, the different error models complicate the use of GPU-accelerated HDC frameworks like OnlineHD [13], which implement only correct computations. Hence, an evaluation is often performed with custom software solutions, which results in very long computation times for the evaluation, limiting the viability for a comprehensive design space exploration.

An FPGA-based HDC implementation offers a significant speedup compared to a pure software evaluation. A complete test run of the MNIST test set, including encoding, takes about 7 ms. The design shown in Fig. 2 is extended with error injection modules. Each module can be configured individually with an error probability and triggers bit flips in the data. Other error models can be implemented as well. Fig. 4 demonstrates the impact of different error rates on the inference accuracy. Errors in the AM are injected during the read of the class hypervectors, the XOR and popcount operation is performed accurately. Similarly, the encoder is affected through errors while reading the item hypervectors. Both modules, AM and IM, demonstrate the robustness of HDC against errors. In contrast, the error injection in the four HP ports impacts the raw sample data. Here, the impact is the highest since a bit flip changes the pixel values leading to a different query hypervector.

4 ANOMALY DETECTION USING HYPERDIMENSIONAL COMPUTING

Anomaly/outlier detection is an everlasting problem across different fields such as IoT and finance. We have developed two HDC-based methods for anomaly detection: HDAD [31] detects anomalies based on reconstruction errors, while ODHD [30] presents a one-class HV-based approach.

4.1 Reconstruction Error-based Anomaly Detection

Inspired by autoencoder-based anomaly detection, which detects anomalies based on reconstruction errors, we develop anomaly detection based on the reconstruction errors of HVs, as shown in 5.

For the anomaly detection pipeline in HDAD, HDAD first encodes training data with normal patterns into HVs then generates a reference HV aggregating all the training HVs representing all the normal patterns in the training set. In the inference stage, HDAD encodes query data following the same encoding strategy, named query HV HV_q. HDAD adds HV_q to the reference HV then reconstructs and decodes the query data from the reference HV. For the purpose of anomaly detection, HDAD computes the reconstruction error, which is the distance between the original input and reconstructed data and utilizes a predefined threshold to filter out the anomalies. The data with distance higher than the threshold will

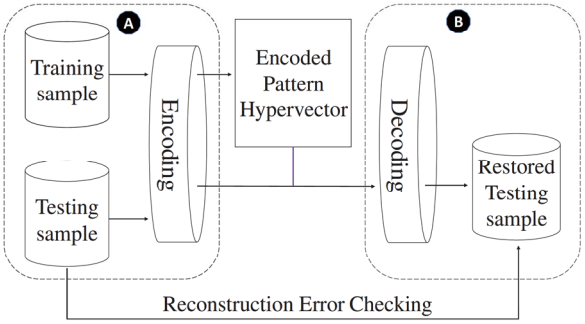

Figure 5: Overview of HDAD process [31]

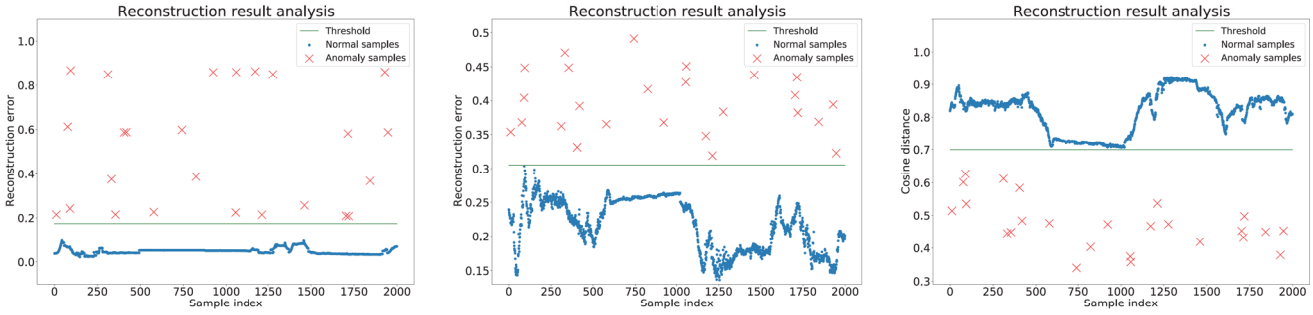

Figure 6: Evaluation result of HDAD anomaly detection for sensor reading data (based on MSE error, MAE error and cosine distance respectively) [31].

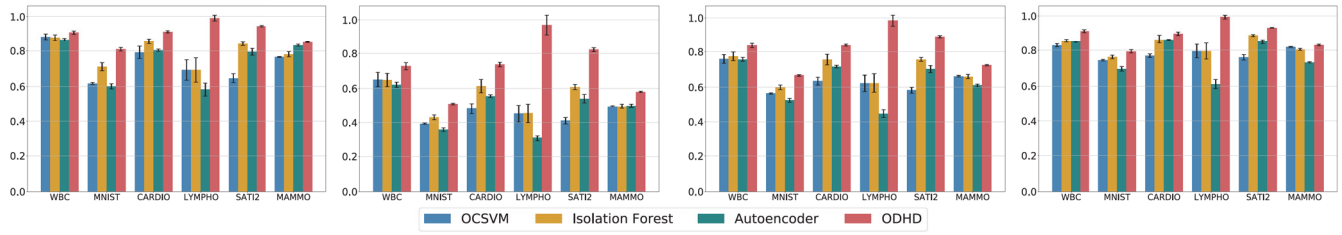

Figure 7: Evaluation result of ODHD outlier detection [30] (based on four metrics: accuracy, average precision, F1-score and ROC-AUC respectively). WBC, MNIST, CARDIO, LYMPHO, SATI2 and MAMMO are the corresponding dataset in [23]

be detected as an abnormal sample. Three different distance metrics are employed in HDAD, including mean squared error (MSE), mean absolute error (MAE) and cosine distance, to determine the reconstruction error.

To evaluate the performance of HDAD, we use the data from the AEGIS Big Data Project [14], which is an autonomous driving dataset containing data from variant types of sensors including bus temperature and pressure sensors, IMU sensors and GPS sensors. The dataset contains continuous generated data from sensors during a driving trip. We use data collected during the first 100 seconds, considered as the normal vehicle states, to formulate a principle pattern for outlier detection on the follow up stage. And we test the HDAD based on the following 100 seconds data with random injected anomalies. We use MSE, MAE and cosine similarity to measure the reconstruction error and to separate the anomalies from the normal data. The results are shown in Fig. 6, where we can see that HDAD achieves 100% accuracy in detecting anomalies using all three metrics.

4.2 One-Class HV-based Anomaly Detection

We also develop a one-class HV based anomaly detection, referred to as ODHD [30], which follows a P-U learning scenario. That is, ODHD is trained with only normal data, and is tested with unseen data samples (can be either normal data or anomalies). The architecture of ODHD is illustrated in 8. In this case, ODHD can extract the normal patterns or representations from ground truth data. By establishing the one-class HV, ODHD learns the standard

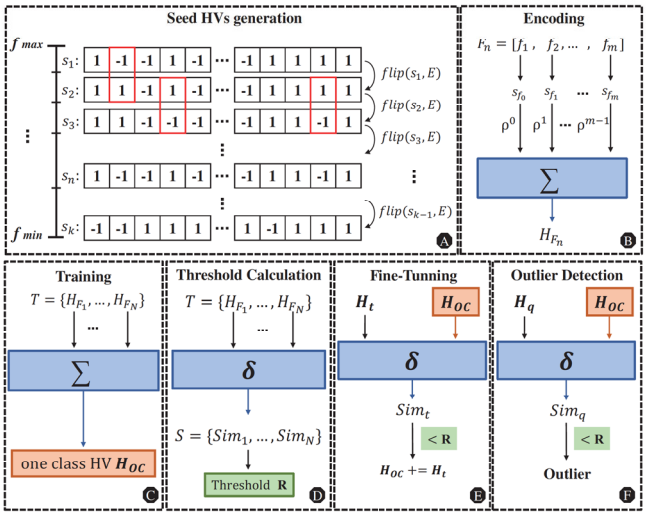

Figure 8: ODHD architecture with six key phases [30].

pattern of normal data, where all the data samples deviated from the standard representation will be recognized as anomalies. By measuring the distance between different HVs, which is also the similarity between different patterns, HDC is capable of finding the anomalies or outliers from the normal patterns.

We evaluate the performance of ODHD on six datasets selected from the Outlier Detection Datasets (ODDS) Library [23] spanning multiple application domains across medical diagnosis, image

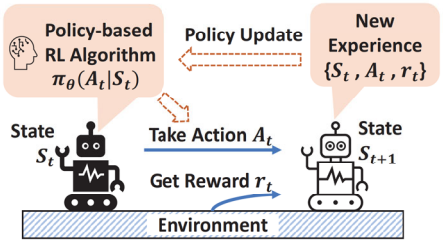

Figure 9: Reinforcement Learning overview.

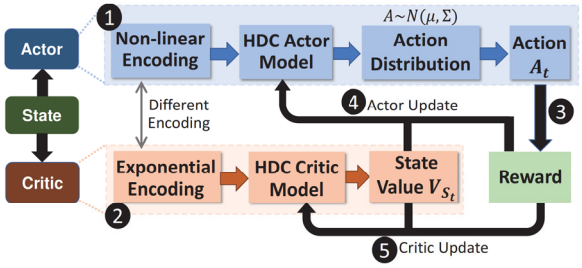

Figure 10: HDPG: hyperdimensional policy-based RL.

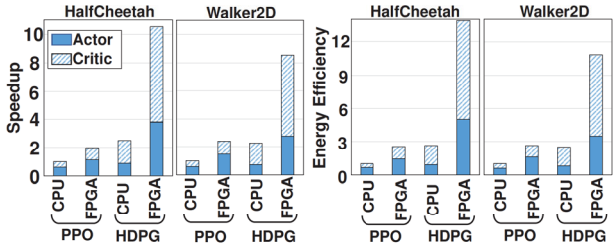

Figure 11: HDPG and PPO performance and energy efficiency on CPU and FPGA platforms (normalized to PPO).

recognition and wireless communication. These datasets are widely used as benchmarks in existing outlier detection studies [1, 24, 38]. Each dataset contains a certain number of outliers specified by the ODDS library. We compare ODHD with baseline models including OCSVM, Isolation Forest (IF) and autoencoder. We employ all the outlier detection models in the same P-U learning format where the evaluation dataset is a mixture of inliers and outliers. In this scenario, we use accuracy, average precision (AP), F1-score and area under receiver operating characteristic curve (ROC-AUC) score to evaluate the performance of ODHD. The experiment results are illustrated in Fig. 7, showing that ODHD outperforms all the baseline models for every dataset in all the metrics.

5 POWERING REINFORCEMENT LEARNING ALGORITHMS WITH HYPERDIMENSIONAL COMPUTING

The latest advancements in Reinforcement Learning (RL) has opened up new opportunities to solve a wide range of complex decision-making tasks. Compared to supervised and unsupervised learning methods, RL does not have direct access to labeled training data. Instead, it trains a self-learning agent using the observations of states and rewards from the environment as shown in Fig. 9 [5]. The learning process of RL is similar to how humans learn to perform new tasks. Initially, the RL agent interacts with the environment and selects an action A_t in a given state S_t with no prior knowledge or past experience. Upon selecting an action, it receives feedback from the environment, called the reward r_t, to update its decision-making process. There are multiple approaches to accomplishing this with the main divide being between model-based and model-free learning. The former trains an agent by simulating the agent's environment in order to predict the system behavior, while the latter involves an agent directly interacting with its environment to accumulate experience for training. The ultimate goal of every RL agent, irrespective of its approach, is to maximize the total returned rewards.

Many of these RL algorithms (e.g., Deep Q-Learning ([19], Policy Proximal Optimization[22]) are implemented using deep neural networks, putting high computational costs on edge devices. The disadvantages of deep learning include a high demand of resources due to the costly back-propagation and gradient-based methods in training[15], extreme sensitivity to noise in data, network, or underlying hardware, as well as lacking human-like cognitive support for long-term memorization and transparency. HDC addresses these challenges, which is why it is advantageous to be used in combination with RL for various tasks. Recent work has demonstrated that HDC is successful in powering RL agents in multiple model-free learning approaches [20, 21], as well as achieving a higher quality of learning (e.g., faster convergence), and improved computational efficiency on multiple devices. In the next sections, we delve into the HDC-based RL algorithms and the applications in which they excel in.

5.1 Policy-Gradient Reinforcement Learning

The first HDC-based RL algorithm developed is HDPG, a model-free policy-gradient algorithm that uses the Actor-Critic framework. This policy-based RL algorithm features an actor for decision-making and a value-based critic for variance reduction, which are both constructed using HDC. At each step, the observed state is encoded to two different hypervectors using a non-linear encoder and an exponential encoder respectively. The actor selects an action and the critic generates a state value using the HDC models. The reward and state values are then used for training both the actor and critic models. The steps of this algorithm are shown in Fig. 10, where it demonstrates how the critic plays an important in updating the actor's policy. We trained this algorithm on multiple Atari games including HalfCheetah and Walker2D and compared it to the Policy Proximal Optimization (PPO) algorithm as shown in Fig. 12. The HDC is able to attain higher rewards, as well as being more energy efficient, which is shown in Fig. 11.

5.2 Value-Based Reinforcement Learning

The latter sub-class of model-free RL that we designed with HDC uses the value-based Q-Learning algorithm. Given an RL environment, this approach approximates the expectation of future rewards, referred to as the *Q-Function*, and relies on these values to inform the agent in its next move. The original Q-Learning algorithm proposed Tabular Q-Learning [32], which could only handle simple environments due to the exponential growth of the state space,

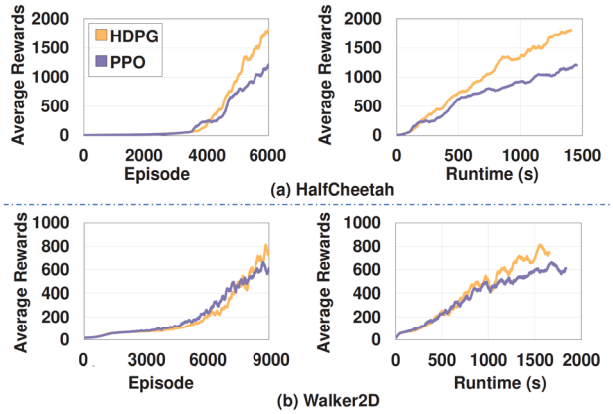

Figure 12: Learning curve comparison between HDPG and PPO for HalfCheetah and Walker2D.

referred to as the *state explosion problem*[2]. However, this implementation does not scale for increasingly more complex state and action spaces, which is why Deep Q-Learning was introduced to handle more complex problems by utilizing a neural network to calculate these values [19].

Analogous to this approach, we use an HDC model in place of a neural network to predict the Q-values. Given a state in the environment, we first encode the state into hyperdimensional space by using an exponential kernel to encode the features of the observation space (i.e., state). The encoded state hypervector is then binded to each *class hypervector* of the HDC model, which represents each encoded action in the environment. This regression model yields a value that represents the Q-value; thus, the action with the largest corresponding Q-value is selected as the next step in the environment. We iterate through this process, collecting feedback (e.g., rewards), from the environment until the episode terminates. Using the Bellman Equation or Dynamic Programming Equation we are able to update the HDC model to compute the error in Q-value prediction and calculate more accurate Q-values [4]. This approach has been tested on multiple Atari game environments, including LunarLander, Acrobot, and Cartpole, to demonstrate a higher learning quality and computational efficiency, compared to DQN [20].

5.3 Solving Network Intrusion Problems

RL has demonstrated its ability in detecting cyber attacks, as well as its potential to recognize new ones. The application of RL to various security problems have been prolific, but most applications have dealt with securing high-powered devices. Thus, the need to extend these systems to resource-constrained edge devices is a necessity henceforth. Harnessing the lightweight, computationally efficiency of HDC, we are able to design an RL Network Intrusion Detection System (NIDS) algorithm that is better equipped to develop robust and more effective security strategies. We demonstrate this by utilizing a modified version of the QHD model to be applied to an abstract Markov game environment for simulating an intrusion detection security problem [12].

The selected cybersecurity environment is modeled as a zero-sum Markov game involving two agents: a cyber attacker and an Intrusion Detection System (IDS). The attacking agent's objective is to compromise a network, while the IDS's opposing task is to secure the computer network and detect any attempted security breaches. The game terminates when either agent completes its defined goal. The environment is implemented to simulate a computer network of interconnected computing devices, where each device has its designated security defense attributes. The attacking agent has a portfolio of actions it can select to compromise a node in the network, while the defending agent selects actions to counter and detect the agent.

Table 1: Comparison of the computational efficiency between the DQN [19] and HDC algorithm on the NVIDIA Jetson Orin I various power settings.

	15W	30W	50W	MAX
DQL	1.9×10^3	1.5×10^3	1.6×10^3	1.0×10^3
HDC	1.4×10^3	1.0×10^3	1.3×10^3	0.8×10^3

Using value-based RL, we implement both agents using HDC to successfully develop robust attack and defense strategies. We test our design on multiple devices, including the NVIDIA Jetson Orin I, to achieve a more effective and efficient solution. The efficiency results, compared to DQN [19] with the Jetson board are presented in Tab. 1 on various low-power settings.

5.4 Optimizing End-Edge-Cloud Networks

The rise of distributed systems and smart devices has made Resource Management an increasingly prevalent orchestration problem to learn to optimize. RL has demonstrated its potential in solving dynamic resource management and scheduling problems [33, 34], which is why we chose to employ HDC RL to solve this type of problem. The algorithm developed, HDHL, is a hybrid learning approach that is used for orchestrating an end-edge-cloud architecture for a deep learning inference task [22, 26]. Hybrid Learning combines model-free and model-based RL to exploit each approach's advantages while diminishing their shortcomings. To further improve the HDC Q-Learning algorithm, we incorporate model-based reinforcement learning to learn a system model and include a planning phase to leverage faster training. This approach is sectioned into three phases: (1) the first phase is used for exploring the network

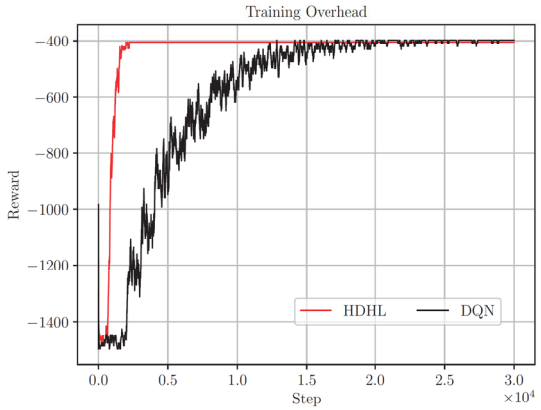

Figure 13: Learning Overhead for DQN and HDHL

environment for data collection, (2) the second for training a system model to learn and simulate the network environment, and (3) a planning phase which involves training the Q-Learning agent using the learned system model from the previous phase.

This hybrid learning approach quickly learns the system's optimal configuration for orchestrating the network, outperforming both the DQN and QHD approaches in terms of computational efficiency and learning quality. HDHL also achieves as much as a 21.0× speedup in the total training and experience time, as well as requiring 4.8× fewer direct interactions with the system environment to learn a near optimal configuration as shown in Fig. 13. As a result, our algorithm significantly speeds up training by reducing the number of real-time interactions with the system and enabling computationally efficient learning.

6 CONCLUSION

Computing beyond the von Neumann era will fundamentally change. Novel technologies will enable new computer architectures. To exploit them most efficiently, new robust algorithms have to be deployed. In this work, we have presented HDC as a promising emerging algorithm. We have investigated hardware implementations for HDC and two applications, namely outlier detection and reinforcement learning. Both areas combined demonstrate that HDC enables computing beyond the von Neumann era.

ACKNOWLEDGMENTS

This research was supported in part by Advantest as part of the Graduate School "Intelligent Methods for Test and Reliability" (GSIMTR) at the University of Stuttgart, National Science Foundation #2202310, #2127780, Semiconductor Research Corporation (SRC), Department of the Navy, Office of Naval Research, and the Air Force Office of Scientific Research.

REFERENCES

[1] Charu C Aggarwal and Saket Sathe. 2015. Theoretical foundations and algorithms for outlier ensembles. *Acm sigkdd explorations newsletter*, 17, 1.
[2] Hooman Alavizadeh, Hootan Alavizadeh, and Julian Jang-Jaccard. 2022. Deep q-learning based reinforcement learning approach for network intrusion detection. *Computers*, 11, 3, 41.
[3] Alon Amid et al. 2020. Chipyard: integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40, 4, 10–21.
[4] Richard Bellman. 1952. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38, 8, 716.
[5] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. 2018. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11, 3-4, 219–354.
[6] Lulu Ge and Keshab K. Parhi. 2020. Classification using hyperdimensional computing: a review. *IEEE Circuits and Systems Magazine*, 20, 2, 30–47.
[7] Paul R Genssler, Hamza E Barkam, Karthik Pandaram, Mohsen Imani, and Hussam Amrouch. 2022. Modeling and predicting transistor aging under workload dependency using machine learning. *arXiv preprint arXiv:2207.04134*.
[8] Paul R. Genssler and Hussam Amrouch. 2022. Brain-inspired computing for circuit reliability characterization. *IEEE Transactions on Computers*.
[9] Paul R. Genssler and Hussam Amrouch. 2021. Brain-inspired computing for wafer map defect pattern classification. In *IEEE International Test Conference*.
[10] Paul R. Genssler, Victor Van Santen, Jörg Henkel, and Hussam Amrouch. 2022. On the reliability of fefet on-chip memory. *IEEE Transactions on Computers*, 71, 4, 947–958.
[11] Paul R. Genssler, Austin Vas, and Hussam Amrouch. 2022. Brain-inspired hyperdimensional computing: how thermal-friendly for edge computing? *IEEE Embedded Systems Letters*.
[12] Kim Hammar and Rolf Stadler. 2020. Finding effective security strategies through reinforcement learning and Self-Play. In *International Conference on Network and Service Management (CNSM 2020) (CNSM 2020)*. Izmir, Turkey.
[13] Alejandro Hernandez-Cane, Namiko Matsumoto, Eric Ping, and Mohsen Imani. 2021. Onlinehd: robust, efficient, and single-pass online learning using hyperdimensional system. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 56–61.
[14] Christian Kaiser, Alexander Stocker, and Andreas Festl. 2019. Automotive CAN bus data: An Example Dataset from the AEGIS Big Data Project. (2019).
[15] Pentti Kanerva. 2009. Hyperdimensional computing: an introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1, 2.
[16] Shubham Kumar, Swetaki Chatterjee, Simon Thomann, Paul R. Genssler, Yogesh S. Chauhan, and Hussam Amrouch. 2022. Cross-layer fefet reliability modeling towards robust hyperdimensional computing. In *IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC'22)*.
[17] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database.
[18] Dongning Ma et al. 2021. Molehd: automated drug discovery using brain-inspired hyperdimensional computing. (2021).
[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
[20] Yang Ni, Danny Abraham, Mariam Issa, Yeseong Kim, Pietro Mercati, and Mohsen Imani. 2022. Qhd: a brain-inspired hyperdimensional reinforcement learning algorithm. (2022).
[21] Yang Ni, Mariam Issa, Danny Abraham, Mahdi Imani, Xunzhao Yin, and Mohsen Imani. 2022. Hdpg: hyperdimensional policy-based reinforcement learning for continuous control. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (DAC '22). Association for Computing Machinery, 1141–1146.
[22] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. 2018. Integrating planning for task-completion dialogue policy learning. *CoRR*, abs/1801.06176.
[23] Shebuti Rayana. 2016. Outlier detection datasets (odds) library. (2016).
[24] Saket Sathe and Charu Aggarwal. 2016. Lodes: local density meets spectral outlier detection. In *SIAM international conference on data mining*.
[25] Gloria Sepanta. Optimal hardware implementation of hyperdimensional computing. MSc thesis, Shahid Bahonar University of Kerman, (2022).
[26] Richard S. Sutton. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*. Morgan Kaufmann, San Francisco (CA), 216–224.
[27] Rahul Thapa, Bikal Lamichhane, Dongning Ma, and Xun Jiao. 2021. Spamhd: memory-efficient text spam detection using brain-inspired hyperdimensional computing. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*.
[28] Simon Thomann, Paul R Genssler, and Hussam Amrouch. 2022. Hw/sw co-design for reliable in-memory brain-inspired hyperdimensional computing. *arXiv preprint arXiv:2202.04789*.
[29] Simon Thomann, Hong Lam Giang Nguyen, Paul R. Genssler, and Hussam Amrouch. 2022. All-in-memory brain-inspired computing using fefet synapses. *Frontiers in Electronics*, 3.
[30] Ruixuan Wang, Xun Jiao, and Sharon Hu. 2022. Odhd: one-class brain-inspired hyperdimensional computing for outlier detection. In *IEEE/ACM Design Automation Conference (DAC)*.
[31] Ruixuan Wang, Fanxin Kong, Hasshi Sudler, and Xun Jiao. 2021. Brief industry paper: hdad: hyperdimensional computing-based anomaly detection for automotive sensor attacks. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 461–464.
[32] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning*, 8, 3, 279–292.
[33] Yue Xu, Hyung Gyu Lee, Yujuan Tan, Yu Wu, Xianzhang Chen, Liang Liang, Lei Qiao, and Duo Liu. 2019. Tumbler: energy efficient task scheduling for dual-channel solar-powered sensor nodes. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 1–6.
[34] Siyu andothers Yue. 2012. Reinforcement learning based dynamic power management with a hybrid power supply. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*.
[35] Sizhe Zhang, Mohsen Imani, and Xun Jiao. 2022. Scalehd: robust brain-inspired hyperdimensional computing via adapative scaling. In *2022 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE.
[36] Sizhe Zhang, Ruixuan Wang, Dongning Ma, Jeff Jun Zhang, Xunzhao Yin, and Xun Jiao. 2022. Energy-efficient brain-inspired hyperdimensional computing using voltage scaling. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 52–55.
[37] Sizhe Zhang, Ruixuan Wang, Jeff Jun Zhang, Abbas Rahimi, and Xun Jiao. 2021. Assessing robustness of hyperdimensional computing against errors in associative memory. In *2021 IEEE 32nd International Conference on Application specific Systems, Architectures and Processors (ASAP)*. IEEE, 211–217.
[38] Arthur Zimek et al. 2013. Subsampling for efficient and effective unsupervised outlier detection ensembles. In *KDD*.