Brain-Inspired Hyperdimensional Computing for Ultra-Efficient Edge AI

Special Session Paper

Hussam Amrouch, Mohsen Imani, Xun Jiao, Yiannis Aloimonos, Cornelia Fermuller, Dehao Yuan, Dongning Ma, Hamza E. Barkam, Paul R. Genssler, Peter Sutor

***Abstract*—Hyperdimensional Computing (HDC) is rapidly emerging as an attractive alternative to traditional deep learning algorithms. Despite the profound success of Deep Neural Networks (DNNs) in many domains, the amount of computational power and storage that they demand during training makes deploying them in edge devices very challenging if not infeasible. This, in turn, inevitably necessitates streaming the data from the edge to the cloud which raises serious concerns when it comes to availability, scalability, security, and privacy. Further, the nature of data that edge devices often receive from sensors is inherently noisy. However, DNN algorithms are very sensitive to noise, which makes accomplishing the required learning tasks with high accuracy immensely difficult. In this paper, we aim at providing a comprehensive overview of the latest advances in HDC. HDC aims at realizing real-time performance and robustness through using strategies that more closely model the human brain. HDC is, in fact, motivated by the observation that the human brain operates on high-dimensional data representations. In HDC, objects are thereby encoded with high-dimensional vectors which have thousands of elements. In this paper, we will discuss the promising robustness of HDC algorithms against noise along with the ability to learn from little data. Further, we will present the outstanding synergy between HDC and beyond von Neumann architectures and how HDC opens doors for efficient learning at the edge due to the ultra-lightweight implementation that it needs, contrary to traditional DNNs.**

***Index Terms*—Hyperdimensional Computing, Embedded Systems, Energy-efficient Computing, Adversarial Attack, Voltage Scaling, In-memory Computing, Security, Graphs, Robotics, Computer Vision**

I. Introduction

THE field of edge AI challenges machine-learning (ML) methods with a broad set of requirements. The inference accuracy is not the only goal, each system design is a tradeoff of various aspects. Due to energy constraints, any algorithm has to be as efficient as possible, simply increasing the layers of a neural network is not sustainable. Furthermore, real-time requirements limit the time available for an inference operation. To achieve an adaptable system, (re)training at the edge is necessary. However, the quality of sensors is low, the environmental conditions can cause errors in the system, e.g., in the memory, or employed emerging technologies are less unreliable yet efficient. The ML algorithm has to be robust against such errors in the computation and noise in the data during (re)training and inference.

Brain-inspired hyperdimensional computing (HDC) is a machine-learning (ML) concept not based on neural networks but large vectors [1]. Such vectors have thousands of dimensions and are called hypervectors. Each hypervector component is independent, creating a unique set of properties to address the challenges of edge AI. First, computations are executed on each component individually, making parallelization easy. Thus, the operations can be completed fast, fulfilling real-time requirements and allowing the system to go back into hibernation. In addition, the independent components not only carry the information but are inherently redundant. The computations do not have to be accurate, for a successful inference a high similarity between two hypervectors is sufficient. Hence, if components of a hypervector are corrupted, e.g., from undervolting the memory, the thousands of remaining correct components ensure a reliable operation. For a similar reason, robustness against noise in the data is high. A noisy data sample is still similar to a noise-free one. After encoding the real-world data into hyperspace, the hypervector representing this sample is again similar. The encoding process does not require any back propagation and thus a few data samples are sufficient to create an HDC model. HDC has been applied to many different areas include circuit reliability [2], drug discovery [3], anomaly and outlier detection [4, 5], Computer Vision and Robotics [6–8], among others [9].

In this paper, HDC for ultra-efficient edge AI is explored from different perspectives. Section II explores software-only implementations and highlights a temperature and energy challenge motivating the need for dedicated hardware and in-memory solutions. In Section III, HDC's robustness against noise is increased by different methods, and the defense against adversarial attacks is described. With an efficient implementation, knowledge is represented in graphs in Section IV enabling reasoning with HDC and recovery of lost memories. Section V describes the concept of hyperdimensional active perception, allowing a system to generate an informative reaction from the environment by intelligently taking an action.

Hussam Amrouch and Paul R. Genssler are with the Chair of Semiconductor Test and Reliability (STAR), University of Stuttgart, Stuttgart, Germany. E-mail: {genssler, amrouch}@iti.uni-stuttgart.de. Mohsen Imani and Hamza E. Barkam are with the Bio-Inspired Architecture and Systems (BIASLab), UC Irvine, California, USA. E-mail: {m.imani, herrahmo}@uci.edu. Xun Jiao and Dongning Ma are with the Dependable, Efficient, and Intelligent Computing Lab (DETAIL), Villanova University, Pennsylvania, USA. E-mail: {xun.jiao, dma2}@villanova.edu. Yiannis Aloimonos, Cornelia Fermuller, Dehao Yuan, and Peter Sutor are with the Perception and Robotics Group, University of Maryland, Maryland, USA. E-mail: yiannis@cs.umd.edu, fer@cfar.umd.edu, dhyuan@umd.edu, psutor@umd.edu.

II. HW/SW Codesign for Efficient Brain-inspired Hyperdimensional In-Memory Computing

HDC is often proposed as a less energy-intensive alternative to neural network-based and traditional ML algorithms. Dedicated hardware accelerators are proposed to further increase the efficiency of the implementations to execute training and inference with HDC. Because the concept of HDC is based on hypervectors with independent components, parallelizing the basic operations is straight forward. For hypervectors with floating point components, the base operations include a component-wise addition and multiplication. Hence, these operations are mathematically simple and do not require sophisticated and complex computations. However, hardware implementations are less flexible than pure software implementations. Changes to the algorithm or a change of the datatype of the hypervectors' components, e.g., a reduction in bit width, can require a revision of the hardware implementation to maximize the energy efficiency. In contrast, software-only solutions are less efficient to execute on the device but can be easily updated. Hence, consumer electronics, especially in fast-paced markets like wearable or mobiles, extensively use software implementations to reduce time to market and save development costs. Only in recent years have mobile SoCs been enhanced by dedicated neural network accelerators, although the ML algorithms have been used much longer. In summary, emerging algorithms, like HDC, cannot be evaluated with dedicated hardware alone, but their software-only metrics have to be considered as well.

A. Power and Temperature Evaluation

For portable battery-powered devices, energy consumption is a major concern. Algorithms have to be efficient to not drain the batteries. In addition, wearable and handheld devices have a much stricter temperature limit. While embedded and automotive devices can dissipate the generated heat into the environment or the surrounding structure, wearables come into contact with the human skin. Hence, high temperatures have to be avoided to not harm the user. Furthermore, wearables are lightweight and thus the devices have less thermal capacity.

In this section, various software-only HDC implementations are compared to neural network-based and traditional ML algorithms [10]. The experiments are executed on a Raspberry Pi 4 Model B to represent an off-the-shelve embedded device without dedicated HDC or ML hardware. The Broadcom 2711 SoC features four ARM Cortex-A72 cores with a nominal frequency of 1500 MHz. If a core temperature of 80 °C is exceeded consistently, the CPU frequency is automatically throttled down to 600 MHz. With an ambient temperature of around 20 °C, the SoC has an idle temperature of 41 °C. As a second metric, the power consumption of the whole board is measured. An INA260 sensor is placed between the output of the power supply and the whole Raspberry Pi board. Idle power consumption is about 3.5 W at 600 MHz.

Various implementations of ML algorithms and datasets are evaluated. First, OnlineHD is based on PyTorch (Python with C backend) and encodes a data sample by multiplying its features with a floating-point base hypervector [11]. The second HDC implementation is HD-Lib [12], which is implemented solely in C and for binary hypervectors. An SVM and a KNN is implemented based on the Scikit-learn library (Python with C backend). A simple CNN is implemented with Tensorflow (Python with C backend), with the following layers: convolutional, max pooling, dropout, flatten, dense, dropout, dense. In total, the CNN has 15,000 trainable parameters. The ISOLET and Fashion-MNIST datasets serve as workloads.

For the ISOLET dataset, the KNN has nine neighbors and the Euclidean distance metric ("Minkowski" with power parameter two). These parameters were found through hyperparameter search and the inference accuracy is 92.1 %. The inference accuracies for OnlineHD are given in Tab. I for various dimensions with retraining. For HD-Lib [12], the samples are quantized to 13 levels providing the highest accuracy, which was also observed in [13]. These 13 level hypervectors are created as a continuous item memory [1]. Each sample is encoded as a record [1] where a randomly generated key hypervector is bound to the corresponding level hypervector and all pairs bundled into a single sample hypervector.

Fig. 1 shows that both HDC-based models cross the soft thermal limit of 80 °C. KNN does not reach this limit because about one third of the inference time is limited to a single CPU core, an implementation detail of the Scikit-learn library. Assuming full CPU utilization, the KNN would process 520 samples/s on a par with the HD-Lib model.

OnlineHD uses mainly floating-point operations whereas, the HD-Lib implementation primarily comprises additions and bit operations with integers. By using a record encoding, the number of HDC operations (binding, bundling, permutation) is lower compared to an n-gram encoding because each feature value is only handled once (compared to n times) and not permuted. If the 3.5 W idle power consumption is subtracted from the mean power, then OnlineHD consumes 15 % more compared to HD-Lib, which correlates with the 12 % higher CPU utilization. Although HD-Lib's record-encoding is efficient, the pure C-implementation is not heavily optimized and is outperformed by the PyTorch-based OnlineHD by 4X for processing speed (2370 samples/s). The temperature for both implementations is at a similar level above the 80 °C soft threshold. In summary, a hand-crafted encoding implemented in C-code does not perform better than the unspecific encoding of OnlineHD with a general-purpose CPU.

Image classification is evaluated on the Fashion MNIST dataset with OnlineHD, a CNN, and an SVM implementation. The inference accuracy with OnlineHD varies from 82.3 % to 86.7 % with a dimension of 500 to 10,000. The CNN achieves a comparable 85.8 % and the SVM performs best with 89.4 %. Fig. 2 shows the impact on the CPU temperature. OnlineHD crosses the soft temperature limit of 80 °C during the inference

TABLE I
Inference accuracy for ISOLET dataset. The samples/s and mean power refer to 4096 bits and 2048 floating-point hypervectors.

Implementation	2048	4096	Samples/s	Mean Power (W)
HD-Lib, binary	83.5	**87.1**	537.7	5.61
OnlineHD, real	**94.7**	95.1	2369.5	5.92
KNN, $n = 9$	– 92.1 –		390.1	5.48

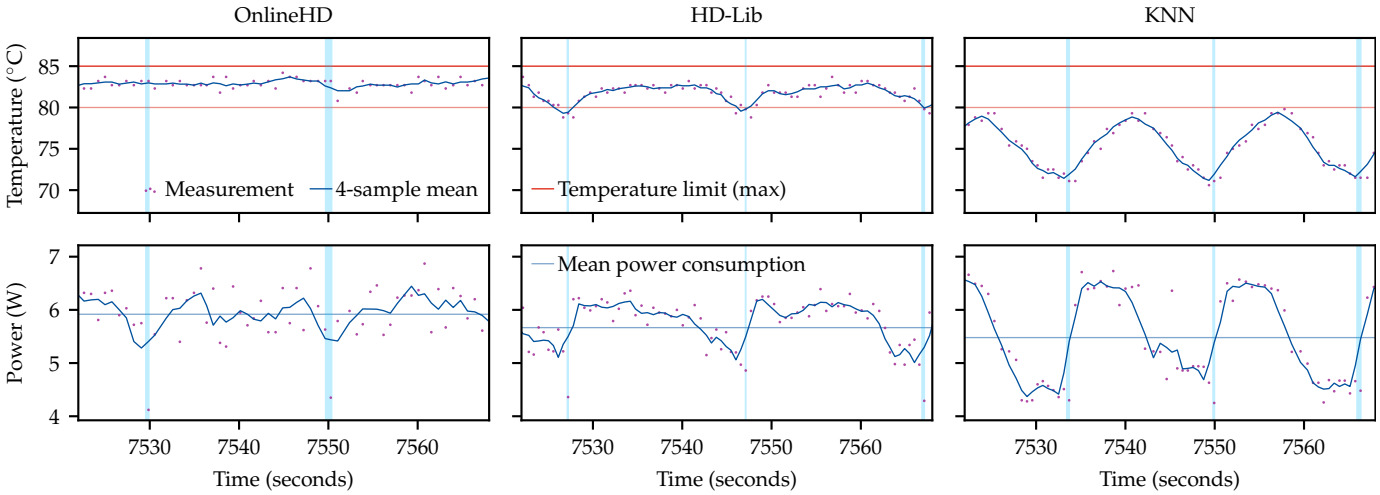

Fig. 1. Temperature and power measurements for HDC and KNN implementations for the ISOLET dataset. While OnlineHD consumes the most power and generates the most heat, it performs 4X and 5X more inferences than HD-Lib and KNN. The light blue vertical intervals indicate a restart of the inference cycle.

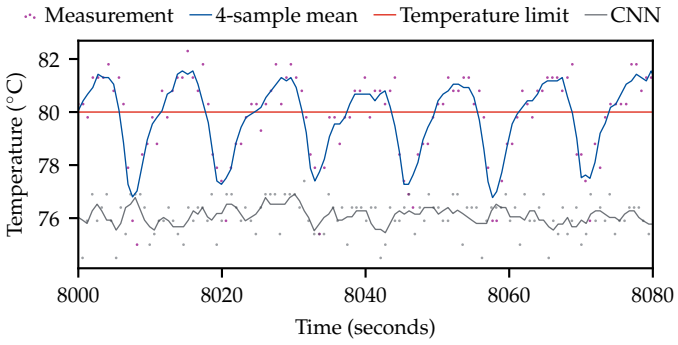

Fig. 2. Comparison of the CNN implementation with OnlineHD for inference on the Fashion MNIST dataset. The CNN processes 2.4X more images per second at a lower temperature although both implementations are based PyTorch.

cycles consistently with an average of 79.7 °C. Yet, it does not push the system close to the 85 °C maximum temperature limit and thus the impact on the CPU frequency is minimal, the SoC does not throttle the CPUs. The temperature drops frequently between two inference cycles. During each complete cycle, the inference is performed on the whole test dataset of 10,000 images and house keeping tasks are performed, like logging the measured metrics and calculating the inference accuracy. The CNN does not see such drops because its associated temperature is comparatively low at a stable 76.1 °C on average. In addition to a lower temperature, the CNN also process 2.4X more images per second, 1000 with OnlineHD compared to the CNN's 2427. Further, power consumption is reduced by 0.2 W.

To simulate a wearable device, the maximum temperature is lowered to 60 °C. Although the SoC throttles the CPU cores to 600 MHz, OnlineHD still exceeds this lowered temperature limit by 6 °C on average. The inference speed drops to 382 images per second. The CNN is less impacted because of its overall less energy-intense computations. In summary, HDC implementations have not yet reached the same maturity-level as neural network-based algorithms, even though both implementations are based on a similar underlying framework. Hence, for ultra-efficient edge AI systems, a hardware/software codesign approach is still required.

B. *Computing-in-Memory for Efficient Inference*

One of the main limits on energy efficiency is the von Neumann bottleneck. It describes the challenge of moving the data from any storage closer to the processing elements. Each data transfer costs orders of magnitude more than an operation on it [14]. Hence, reducing the movement of data within a computing system increases its energy efficiency. One approach is computing-in-memory, in which the computations happen where the data already resides, inside the memory. With HDC for edge AI, inference is most likely the most frequent operation. For binary hypervectors, the search in the associate memory (AM) involves the computation of the Hamming distance from all stored class hypervectors with the query hypervector. In a traditional system, each class hypervector has to be transferred through the cache hierarchy to the CPU. With a computing-in-memory approach, to avoid expensive data transfers, the structure of the memory is changed to compute the Hamming distance directly in the AM [15].

Each binary class vector is split into small blocks of, e.g., $n = 15$ bits [16]. Each of these blocks comprises n TCAM cells, each storing a single bit. All TCAM cells within a block are connected to a match line, which is precharged to the supply voltage before an inference operation. Then, the corresponding bits of the query vector are applied to the block's bits. If the stored and the query bit do not match, the TCAM cell forms a conductive path discharging the match line. The more TCAM cells do not match (query and class hypervector are dissimilar), the faster the match line is discharged. A sense amplifier maps the discharge time to the number of mismatching bits. To obtain the Hamming distance of two full hypervectors, the number of mismatching bits are collected from all blocks.

An AM for c classes and hypervectors of dimension D consists of $c * (D/n)$ blocks. During an inference operations, the above-described procedure is executed in all blocks at the same time, effectively parallelizing the AM lookup. The individual Hamming distances are accumulated and the class hypervector with the lowest Hamming distance is the inference result. All TCAM cells within such an AM can be implemented with conventional CMOS technology or with emerging FeFET technology. The following results are based on SPICE models calibrated to reproduce Intel's 14 nm FinFET measurements [17].

Both technologies, traditional CMOS and the emerging FeFET technology, are impacted by process variation, resulting in blocks reporting an incorrect Hamming distances. However, the impact on FeFET is greater, doubling the error probability (78 %) compared to SRAM (40 %) for a block size of 15 bits and 0.5 V supply voltage. Nevertheless, HDC is robust against errors in the reported Hamming distance. For language classification and a block size of 7 bits, the inference accuracy drops by 0.2 % and 0.8 % for SRAM and FeFET, respectively. The major advantage of FeFET over SRAM is, besides its non-volatility, that TCAM cells required 8X fewer transistors and thus less chip area. This additional area can be invested into multiple instances of the AM to reduce the impact of process variation by averaging the results. With five replicas, the inference accuracy drop is limited to 0.3 % and close to the level of the mature SRAM technology. One Hamming distance computation with SRAM-based TCAM cells consumes 0.25 pJ to 0.5 pJ, depending on the various design parameters. A FeFET-based AM requires similar amounts of energy but can be turned off during hibernation whereas the SRAM has to be powered all the time.

III. Robust Hyperdimensional Computing Against Hardware Errors and Attacks

A. Hardware Errors

1) Error Characterization: The major advantages of HDC over conventional machine learning algorithm are higher energy efficiency and more compact model. Therefore, the primary targets of HDC deployment naturally become resource constrained platforms such as edge and embedded architectures, where the energy efficiency usually comes at the expense of higher degrees of hardware error because of the compromised "near-perfect execution". While research has focused on evaluating the conventional machine learning algorithms about the robustness against such hardware errors, there has not been an established line of literature for the emerging HDC. Our works attempt to present insights into the effect of hardware errors on HDC model, targeting at two major sources: memory errors and voltage scaling induced errors.

The processing of HDC requires extensive support from memories. During encoding, item memory is frequently accessed to fetch the corresponding item HVs of each feature value; during training and inference, the associative memory is also subject to numerous reading and writing operations so that an HDC model can be trained and then evaluated.

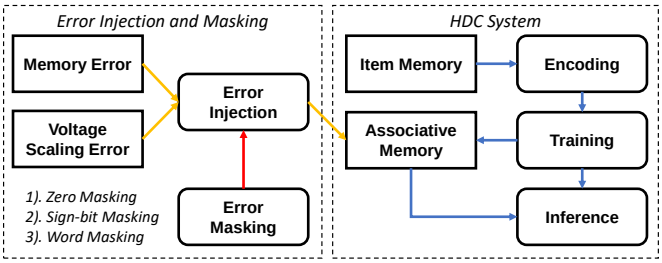

Fig. 3. Framework for hardware error injection and masking for HDC system.

However, the manufacture process and device aging and wear-out effects such as negative bias temperature instability (NBTI), electromigration, and time-dependent dielectric breakdown can cause memory to exhibit error patterns. To systematically assess the error schemes and their influence, we measure, quantify, and characterize the impact of errors in associative memory on the classification accuracy of HDC models [18].

We select the most widely used single bit flip (SBF) error model to quantify the memory error: once an error occurs, the corresponding bit is flipped (becomes corrupted). The error occurs at the associative memory when the HDC model is making an inference with a predefined possibility referred to as error rate. We sweep the error rate from 10^{-9} to 10^{-1}, the HV dimensions of 10,000, 5000 and 3000, and the data widths of 16-bit, 8-bit and 1-bit. Results on three applications show that HDC possesses strong robustness against memory errors that, without any error mitigation schemes, error rate up to 10^{-6} will not induce any noticeable accuracy loss. As to dimensions and data widths, in general lower dimensions and data widths can tolerate higher error because the value shifts caused by bit errors have less impact on them.

In addition to memory errors, voltage scaling is also a major source of errors for edge intelligence architectures. Voltage scaling is one important technique for low power designs to achieve higher energy efficiency at the expense of higher latency and potential timing violations, which can subsequently translate into accuracy degradation at the application level.

To analyze the impact of voltage scaling during HDC inference, we target voltage-scaled on-chip SRAMs fabricated in FDX22 (22nm) technology [19]. We first profile the bit error rate under different voltages from the nominal 0.8 V to as aggressive as 0.35 V and then inject errors for the HDC inference process based on the rates profiled. Results on four datasets show that reducing voltage from 0.8 V to 0.6 V will not trigger notable accuracy loss, thus we refer to the region between 0.8 V and 0.6 V as the safe region. However, more aggressive voltage scaling will bring drastic accuracy loss, e.g., when the voltage lowers to 0.5 V, the accuracy plummets to only 40 % which is about 50 % lower compared with baselines without voltage scaling.

2) Robustness Enhancement: We propose three low cost error mitigation or protection schemes: zero masking, sign-bit masking and word masking as illustrated in Tab. II to ameliorate the degraded accuracy induced by memory errors or voltage scaling [18, 19]. Zero masking and sign-bit masking set all the corrupted bits (marked as X) to 0 and the sign-bit respectively,

TABLE II
MASKING SCHEMES FOR PROTECTING HDC MODELS FROM HARDWARE ERRORS

	Zero Masking			Sign-bit Masking			Word Masking		
error-free	11010010	00111001	11110011	11010010	00111001	11110011	11010010	00111001	11110011
before masking	11X100X0	0X111X01	11110011	11X100X0	0X111X01	11110011	11X100X0	0X111X01	11110011
after masking	11010000	00111001	11110011	11110010	00111001	11110011	00000000	00000000	11110011

while word masking sets the entire number (numerical value) to 0.

Using masking schemes, we can fortify the robustness of HDC models so that error rate up to 10^{-5} and even 10^{-4} for some applications can be tolerated with only 1 % accuracy loss. This is more than one order of magnitude robustness than baseline implementations (without any error masking scheme). As to voltage scaled HDC systems, we can further recovers the accuracy at 0.5 V and can push the voltage even lower to 0.45 V with negligible accuracy loss only. Note that this enables the HDC model to tolerate 10,000 times higher error rate, which can further expands the safe region from 0.8 V–0.6 V to 0.8 V–0.45 V. For energy efficiency, using the masking schemes can bring about 50 % to 70 % energy saving on average.

B. Attacks

1) Attack Characterization: Although machine learning algorithms can achieve comparable or even surpassing accuracy than human for certain applications, they are known to be vulnerable to adversarial attacks. For neural networks, research has shown that by adding negligible perturbation to the original input samples, adversarial samples can be fabricated which can induce inconsistent or incorrect behaviors of the model under attack, posing a substantial threat for security sensitive applications [20]. Although adversarial attacks exist in HDC models as the example in Fig. 4, how they can be automatically generated is not straightforward for HDC. That is, adversarial attacks can be generated using gradient based methods for neural networks, however, as HDC does not primarily own such necessary mathematical properties, gradient-based methods cannot be directly borrowed here and applied.

On the other hand, the emerging coverage-guided fuzz testing which originates from software testing communities is increasingly applied for enhancing the robustness of machine learning algorithms [21]. Fuzz testing is a flexible and heuristic-inspired method which does not require tedious manual labeling of samples and can be applied with black, gray and white-box schemes. We introduce this concept into the HDC domain and develop differential (fuzz) testing frameworks.

For black-box scenarios, we develop HDXplore, a highly automated testing tool specifically designed for HDC based on differential testing [22]. HDXplore takes the original input image without necessarily knowing its label. A set of mutation algorithms are then applied to create perturbed input. The perturbed input along with the original input are fed into the HDC model together for predictions. If the labels are inconsistent, an adversarial sample is successfully generated. Otherwise, HDXplore will iterate the generation process until success. For gray-box scenarios, we propose HDTest as illustrated in Fig. 5, which is also a differential fuzzing testing framework but is coverage-guided [23]. Different from HDXplore which assumes the entire HDC model as a black-box, HDTest has the access to the associative memory and the similarity information of each prediction. Therefore when generating new perturbed samples, the similarity information can be used as a coverage to select higher quality generations which are more likely to induce inconsistency in HDC inference, thus increasing the successful rate of adversarial attacks.

Data poisoning attack is also a realistic threat to machine learning models where noises are injected into labels of training data to impair the performance of ML models [24]. To characterize the impact of such attack on HDC, we propose PoisonHD [25], a poison attack framework that maximizes its effectiveness in degrading the HDC classification accuracy. PoisonHD particularly leverages the internal structural information of HDC models of confidence of each sample and performs label flipping on the most vulnerable samples by their confidence ranking. By flipping 15 % of the total training samples, we are able to lower the model classification accuracy by up to 30 % on three datasets: MNIST 1–7, Dog-Fish, and Breast Cancer.

Fig. 4. Perturbation on just several pixels can "fool" an HDC model to mis-classify 9 as 4.

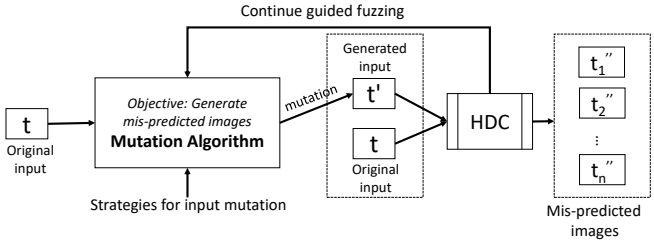

Fig. 5. Coverage-guided differential fuzz testing for HDC

2) Robustness Enhancement: To enhance the robustness of HDC against adversarial attacks, we leverage the generated adversarial samples to retrain the HDC models. Experimental results show that with adversarial retraining, the HDC model can successfully defend up to 30 % of unseen incoming adversarial input samples, which can drastically enhance the robustness of the models against attacks.

To defend data poisoning attacks, we propose a sanitizing defense mechanism which is also referred to as "Oracle Defense" [24]. We train an outlier detection model based on a separate verification set which is not accessible to the attackers. The outlier detection model is also an HDC model but focuses on checking the similarity between the HV and the class that its corresponding input label refers to. If there is a discrepancy between the oracle and the provided label, then this sample is regarded as a "noxious" sample and thus discarded. By using this oracle defense method, we are able to reduce the accuracy loss of PoisonHD attack from up to 30 % to less than 3 %.

The defense mechanisms do come with expenses, for example, although adversarial retraining can reduce the impact caused by attacks, it at the same time poses negative influence on the performance of classifying benign samples, resulting in accuracy loss. In the future, we aim to develop new defense and robustness enhancement algorithms to minimize the impact to the original HDC model.

IV. An HDC Platform for Robust and Efficient Perception and Decision Making at the Edge

As neuroscientists have already shown, the human brain memorizes events as a sparse memory graph [26, 27], where nodes are the objects/events, and the edges represent the correlation between them. The brain does reasoning and analogy by referring to this memory as prior knowledge. For example, as humans, when we see a set of events or objects repeatedly occurring together, these objects get a higher correlation in our graph memory. By referring to this memory, we can identify the correlated objects, make better decisions, and reason about them [28].

Although building up this graph is often easy, the main challenges are: (1) how to effectively represent this graph to enable highly efficient and robust brain-like memorization, and (2) how to perform information retrieval and reasoning on the graph. Unlike the existing graph processing algorithms that perform costly exact computations, brain memorization and cognitive tasks are highly approximate and efficient [29].

We propose GrapHD, a hyperdimensional graph memory that enables robust, efficient, and holographic cognitive learning. Fig. 6 shows an overview of GrapHD. GrapHD encodes various graph data into high-dimensional space (Fig. 6a). The encoding is based on a well-defined set of mathematics. Our encoding represents a graph using a single hypervector, where each dimension represents a neuron. GrapHD enables a wide range of cognitive operations directly over the graph hypervector (Fig. 6b). These cognitive operations extract information from the graph without explicit access to original nodes. We exploit these functionalities to enable several applications, including graph matching, shortest path, and object detection (Fig. 6c).

A. Hyperdimensional Graph Representation

We exploit hyperdimensional mathematics to spread the graph information across the fully holistic high-dimensional representation. In this representation, no hypervector element is more responsible for storing any piece of information than another. Fig. 7 shows the functionality of GrapHD encoding

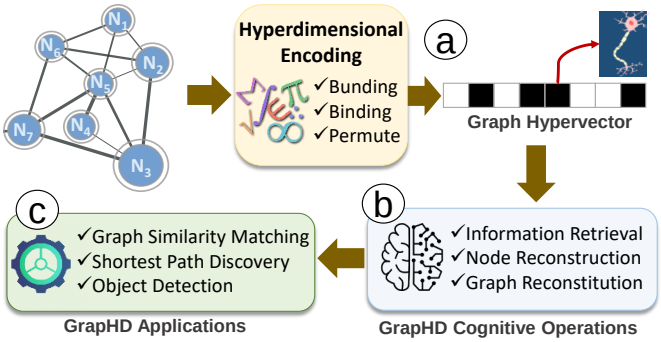

Fig. 6. GrapHD Overview: (a) hyperdimensional graph encoding into a hypervector, (b) GrapHD cognitive operations, and (c) GrapHD applications.

representing unweighted graphs. We first assign a random hypervector $\vec{H}_i$ to each node in the graph (Fig. 7a). Assuming a graph with n nodes, we generate $\{\vec{H}_1, \vec{H}_2, \cdots, \vec{H}_n\}$ as high-dimensional signature of nodes, where $\vec{H}_i$ is a $D-$dimensional vector whose components are randomly chosen from the set $\{-1, +1\}$. Due to random generation, the node hypervectors are nearly orthogonal: $\delta(\vec{\mathcal{H}}_k, \vec{\mathcal{H}}_l) \simeq 0$ $(k \neq l)$, where δ denotes the cosine similarity.

We exploit the node hypervectors to create a memory for each node. The node memory needs to remember all connections that a particular node has to its neighbors (Fig. 7b). For example, we construct the node i memory by accumulating all node hypervectors connected to it: $\vec{M}_i = \sum_j \vec{H}_j$, where j represents all the neighbors of node i. Thanks to HD computing mathematics, the bundling keeps the information of all connections. For example, we can check if memory node i has connection to node k using: $\delta(\vec{\mathcal{M}}_i, \vec{\mathcal{H}}_k)$, where $\delta \gg 0$ and $\delta \simeq 0$ show existence and non-existence, respectively.

After generating a memory for each node, we construct a single hypervector representing a graph. The graph memory should memorize the information of nodes and their connections. To this end, for each node, we associate the node and memory hypervectors, e.g., $\vec{H}_i \oplus \vec{M}_i$ for node i. The bundling of all associated hypervectors generates a graph memory (Fig. 7c):

$$\begin{aligned}\vec{G} &= \frac{1}{2}\left(\vec{H}_1 \oplus \vec{M}_1 + \vec{H}_2 \oplus \vec{M}_2 + \cdots + \vec{H}_n \oplus \vec{M}_n\right) \\ &= \frac{1}{2}\sum_i^n \vec{H}_i \oplus \vec{M}_i\end{aligned}$$

where the graph memory is a compressed, invertible, and transparent model. Given the graph memory $\vec{G}$, we can reconstruct a local node memory using:

$$\vec{H}_i \oplus \vec{G} = \vec{M}_i + noise \approx \vec{M}_i$$

where this approximate equality holds true because the HD vectors are randomly constructed; thus, they are nearly orthogonal. Once we have the local memory, we can check if nodes j and i are connected by calculating the similarity $R = \delta(\vec{H}_j, \vec{M}_i)$, where R is termed as the decision score. If there exist an edge between i and j, then $R \sim 1$. Otherwise, $R \sim 0$.

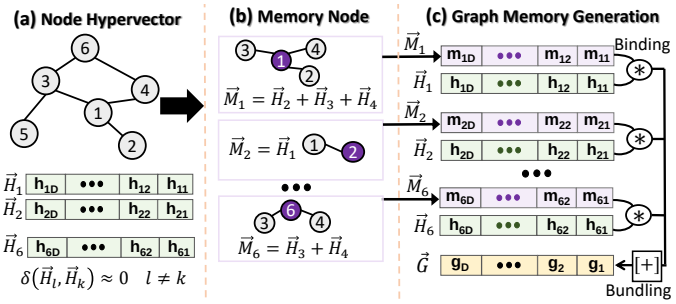

Fig. 7. Graph-memory encoding in GrapHD: (a) node hypervector generation, (b) creating a memory node, and (c) graph memory generation.

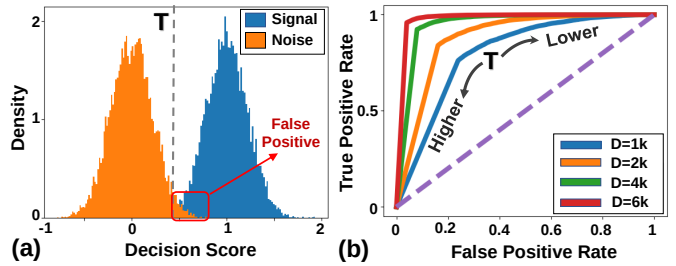

Fig. 8. (a) Distribution of signal and noise during information retrieval, and (b) ROC curves for different dimensionalities.

B. Algorithms with GrapHD Representation

We perform several important cognitive functionalities over the memory graph to extract information or reason based on that. We discuss a few key capabilities which have a wide range of applications in robotic, genomics, signal processing, and machine learning. All tasks can be directly implemented over a single graph memory hypervector, with no need to store original nodes or their connection. In other words, we will show how a single graph hypervector can answer several cognitive questions in a fast and efficient way.

Information Retrieval: The main objective of information retrieval is to extract information about the edges connected to a node and the information associated with each node. We devise a statistical framework to study the errors and data recovery. Given the main memory $\vec{G}$, we can use this to reconstruct the node memory. Using the node memory, we run inference to find the two main quantities – the nodes that share an edge with the current node and the information that has been associated with the current node via binding.

Fig. 8a shows the distribution of signal and noise when an exist and do not exist in reference, respectively. As results indicate, both signal and noise follow Gaussian distribution, where the spread is an effect of interference noise. To identify the existence of a pattern, our goal is to put a threshold that can separate signal and noise distribution. Fig. 8b shows the ROC curve indicating the impact of threshold value on true and false-positive rates. Ideally, we want the ROC curve to pass through the left-top corner, where true and false positive rates are 100 % and 0 %, respectively. The sharp turning point would represent the optimal scenario. However, the ROC would be less sharp if we decrease the dimensionality. For example, in $D = 1k$, signal and noise will have wider distribution; thus, the perfect true positive rate can only be obtained with a very high false-positive rate.

Node Memory Reconstruction: We develop an iterative method to recover the node memory from the graph hypervector in an error-correcting way. The main idea is to first formulate a reasonable estimation of all node memories using the unbinding procedure. Then, we find a revised estimate for all the nodes by recursively canceling out the interference noise.

Fig. 9a shows the impact of hypervector dimensionality and the number of edges on the quality of information retrieval. Our results indicate a larger graph requires higher hypervector dimensionality to ensure full graph memorization. For example, a graph with 100 and 200 edges can be accurately stored in a graph hypervector with $D = 4k$ and $D = 6k$ dimensionality, respectively. Fig. 9b shows the number of required iterations for data recovery. Our technique requires fewer iterations of noise cancellation when the dimensionality of a hypervector is larger than the number of edges that it can accurately store. From another hand, when the dimensionality is much lower than the required value, our algorithm may still require a few iterations, but it would converge to a random solution. In summary, maximum iterations are required when the dimensionality is the lowest possible value that provides enough capacity to accurately recover the stored information.

Fig. 9c also shows the number of mismatched edges during different noise cancellation iterations. Initially, our graph reconstruction comes with a large number of mismatched edges. This mismatch is larger for larger graph sizes. The error rate starts decreasing during our recursive error correction mechanism. When the size of the graph is within a capacity of a hypervector ($n \leq 150$ for $D = 4k$, as shown in Fig. 9a), our reconstruction will accurately recover the model. However, when the hypervector stores more patterns, our data recovery often diverges to a random graph (red line shown in Fig. 9c).

One of the main advantages of hyperdimensional representation is its high robustness to noise and failure [11, 30]. In GrapHD, hypervectors are random and holographic with i.i.d. components. Each hypervector stores the information across all its components so that no component is more responsible for storing any piece of information than another. This makes a hypervector robust against errors in its components. Fig. 9d shows the impact of noise in dimensions on graph memory reconstruction. The results are reported when different percentages of hypervector dimensions are randomly dropped. Our representation provides inherent robustness to such noise, as the data can still be reconstructed when the dimensionality is large enough. For example, our method tolerates 10 % random noise using $D = 6k$ dimensions to represent a graph.

V. Hyperdimensional Active Perception

Artificial Intelligence has undergone impressive leaps in the past couple decades, giving birth to countless networks, models, learning algorithms, architectures, and so on. For virtually every dataset, we now have armies of trained AIs that attempt to solve the problems in different ways, with different strengths and weaknesses. Can we use our existing models to bootstrap AIs for more general tasks? Directly related to this issue, the field of Robotics itself is shifting towards tackling the problem

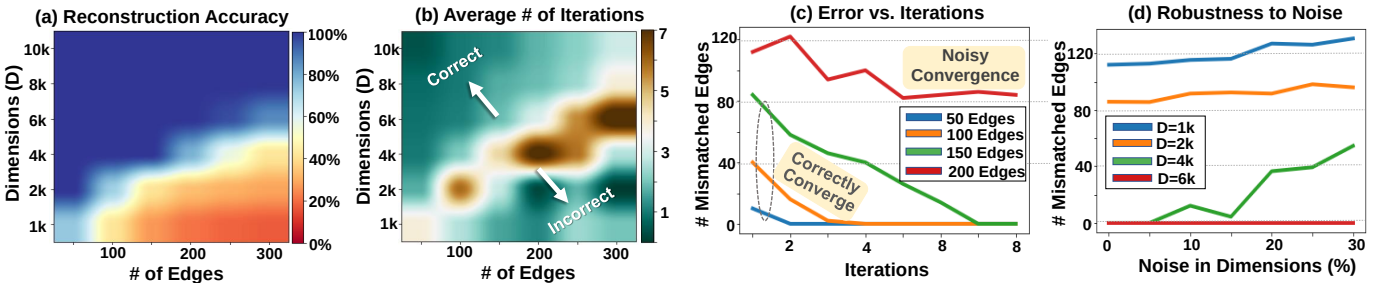

Fig. 9. Memory graph: (a) reconstruction accuracy, (b) required iterations vs. graph size and dimensions, (c) error rate v. iterations, (d) robustness to noise

of integration to support ever more complex sensors and motors in modern robots, utilizing them to solve general problems.

In this section, we explore how HDC models could be applied to the problem of Active Perception in robotics - the notion that an autonomous agent must actively interact with its environment to generate perception signals upon which it then further reacts. This is simply referred to as Hyperdimensional Active Perception (HAP). Furthermore, we explore how HDC's properties can be used to facilitate a type of central AI, which can be used to also solve the problem of integration in robots.

A. *Learning Sensorimotor Control with HAP*

First, we discuss a case study in applying HDC to Active Perception, a move towards HAP in robots [6]. Consider the seemingly straightforward task of teaching an autonomous agent to estimate its ego-motion from neuromorphic visual sensors. In event-based cameras, each "pixel" of the camera asynchronously detects changes in intensity of light, generating so-called events at particular timestamps for that pixel. So, the agent's movements can directly generate perception in the form of events, implying the feature of ego-motion is immediately present in the event stream generated. This gives rise to an active perception situation.

The pipeline of a HAP system for this task is shown in Fig. 10, where an agent moves and generates velocity vectors, shown in (a), as well as raw events from the event camera, shown in (b). The problem of neuromorphic ego-motion detection is easily translated into a hyperdimensional formulation by using motion aligned projections referred to as "time slices" [31], which bin events in time into convenient RGB images that encode the events as blurred motion, shown in (c). Such images are then easily encodeable into binary hypervectors by encoding each pixel intensity and its position by the method described in [6], using representations for intensity and position that can be derived from methods such as those shown in [32]. The black and white square visualizes this binary hypervector in (f). Simply aggregating some of these time slice hypervectors in a sequence encodes a moment in time that the autonomous agent is perceiving, shown in (d). Likewise, on the control side, the ego-motion is a simple 3D vector, encoded via similar methods to the intensity, albeit with finer bins for the range of velocities the agent has experienced in each component. Given a moment of perception hypervector P, and a velocity bin as a hypervector in some component V, the action and perception can be easily bound together by $P \oplus V$, where $\oplus$ is the XOR operation, as shown in (e).

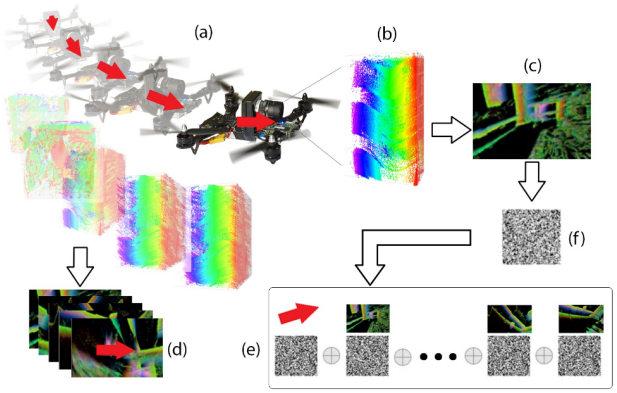

Fig. 10. The pipeline of the HAP system described in [6]. An autonomous agent generates velocities through movement (a) and events in the event camera as it moves (b). The latter are binned into time slices (c). Time slices are paired with their respective velocities across a period of time (d). The hypervector representations of a time slice and a velocity bin are bound together, and then aggregated to represent a moment of the agent's experience (e). Time slice hypervectors are obtained via the RGB image by a pixel wise intensity-location encoding scheme into a single hypervector.

With the perception and control sides encoded and bound as hypervectors, a dataset for HAP is formed. Now, a learning mechanism is required to be able to recall the correct action to perform (the velocity) given a change in perception (the event stream). Both classification and regression methods exist, but classification is focused on for simplicity. Given a moment in time as a hypervector, we want to recall the most likely velocity bin in each dimension. A learning mechanism for this is shown in Fig. 11. During training, a memory hypervector $M = (A_1 + ...) \oplus V_1 + ... + (Z_1 + ...) \oplus V_n$ is formed of velocity bins V_i as classes, bound to training examples for each velocity. Given a new input I, we can classify the velocity by computing $M \oplus I$ and finding the closest velocity hypervector.

This simple mechanism performs surprisingly well. Benchmarking was performed on the MVSEC dataset [33], in which an autonomous car with an event camera navigates an environment. With competitive results to a CNN based approach, the HAP approach requires just one hypervector for M, and n hypervectors for the velocity representations. Inference is blindingly quick, hundreds of inferences per second, requiring a handful of XOR operations. Meanwhile, the training converges within seeing only a fraction of the dataset, approximately 15 seconds of data, requiring approximately 1 second to train on a single CPU core. Effectively, the training

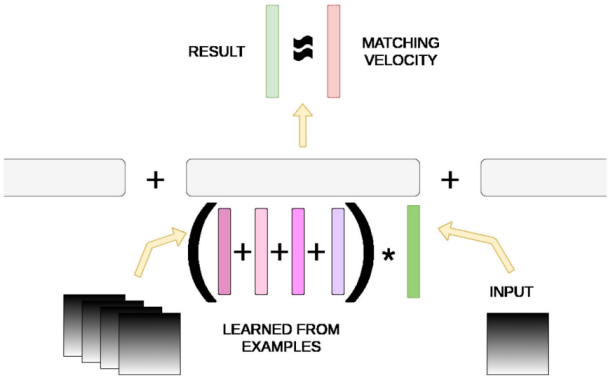

Fig. 11. The Hyperdimensional Inference Layer (HIL). A simple, common mechanism for learning to classify a hyperdimensional vector output, given a hyperdimensional vector input. During training, we aggregate each time slice bundle hypervector (shown in shades of red) that experiences a particular velocity, and bind it to the hypervector representation of this velocity (shown as green). We then aggregate the result for every velocity into a single, memory hypervector M. At testing time, given a new input I, we XOR this with M, and probe each velocity V_i; the one the closest Hamming Distance to $M \oplus I$ is our prediction. This paradigm is generalizeable for arbitrary input and output hypervectors.

can be performed online, in real time, and the small size and rapid inference of the HAP model allows for multiple models to be formed. This is a simple, but powerful demonstration of the potential of HAP, and HDC in robotics, in general.

B. Symbolic Representation and Learning With HDC

For HDC to enable effective integration, a mechanism for integrating multiple sensors, perceptions, and actions is desirable. By projecting symbolic data into a common hyperdimensional space, fusion of different modalities is easily achieved. Suppose we have multiple models that solve a task or provide feature vectors: we would like to fuse their output at a symbolic level via HDC, to form a single predictive model. Another case study shows that this can be readily achieved with HDC [7]. Given several models, memory vectors are formed for each model in a similar fashion to [6], by forming an HIL much like in Fig. 11 for each model as it learns or is presented input after training, and then aggregating these HILs into a single memory that predicts in consensus.

Namely, three different CNN based Hashing Networks were used to simulate differing models: the Deep Quantization Network (DQN), Deep Cauchy Hashing (DCH), and Deep Triplet Quantization (DTQ). Each improves on the previous in performs, and each uses very learning strategies. However, by projecting hashes generated by such networks into hyper-dimensional lengths, HILs were easily learned for each, and then aggregated into a consensus structure. The HIL learns much quicker than the networks themselves, as HDC is well suited to few-shot problems, and project to hyperdimensional lengths eliminated the need to search for optimal hash-length difference. Fig. 12 shows this phenomena for DQN. When all 3 networks were aggregated together, the consensus model outperformed the best performing standalone model (DTQ) by an impressive near 10 % relative improvement, from about 72 % to 92 %. Combined with HAP, these results imply a valid

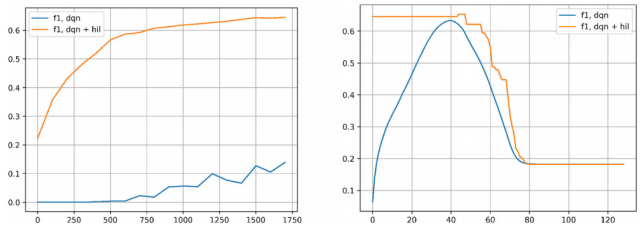

Fig. 12. F1 Score of DQN (blue) vs. DQN + HIL (yellow) over training epochs (left) and Hamming Distances for classification (right).

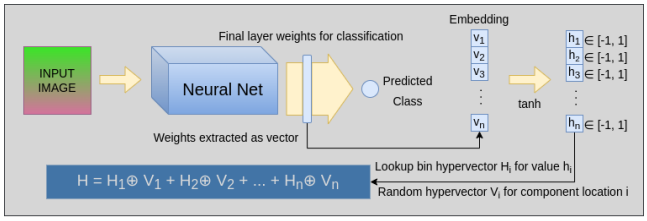

Fig. 13. Given a network and an input, output signals from just before the classifying layer are vectorized and symbolically converted to hypervectors. Applying $\tanh$ normalizes these values to a range of $[-1, 1]$, which is then binned to the desired precision.

strategy for both having Active Perception and a form of AGI for robots for arbitrary sensors and controls.

C. HD-Glue

We can push the idea of symbolic integration in [7] to a further logical conclusion: feature level learning as opposed to the symbolic representations. Given an arbitrary set of networks and in input for them, we can directly encode the vector of output signals just before the final layer of each network. This vector is used by the network to predict the class; we can directly correlate these signals to our output class using HDC. Using the same idea in [7], one can effectively glue neural networks / feature vectors together into a consensus model that outperforms its constituents. This process is referred to as HD-Glue [8]. In Fig. 13, the pipeline of this process is illustrated.

The results indicate that for diverse networks architectures, HD-Glue will form an effective consensus that has all the advantages of HDC. Benchmarking on MNIST and CIFAR-10 and CIFAR-100, against various common techniques for aggregating feature vectors, HD-Glue consistently outperforms, while still being able to handle any length of feature vector. Like with any HIL based approach, new networks can be added, the model is tolerant to any missing networks, and can be trained with very little overhead when compared to the constituent networks. From an integration perspective, this approach is very attractive for integrating existing, trained models for tasks, while still being flexible to handle any new modalities that could be added later. From a robotics perspective, this is ideal to handle the integration problem while still facilitating HAP all the way from the sensor level, to the control level, to the central AI level.

VI. Conclusion

This work provides a comprehensive overview of the latest advances in HDC. Temperature and energy challenges from software-only implementations are explored, motivating the need for dedicated hardware and in-memory solutions. In addition, methods are described to increase HDC's robustness against noise and defend against adversarial attacks. With an efficient implementation, knowledge is represented in graphs enabling reasoning with HDC and recovery of lost memories. Last, hyperdimensional active perceptions is described allowing a system to intelligently take actions to trigger a reaction from the environment.

Acknowledgment

This research was partially supported by Advantest as part of the Graduate School "Intelligent Methods for Test and Reliability" (GS-IMTR) at the University of Stuttgart and in part by the German Research Foundation (DFG) "ACCROSS: Approximate Computing aCROss the System Stack". This study was also supported by National Science Foundation (NSF) #2028889, #2127780, Semiconductor Research Corporation (SRC) Task #2988.001, Department of the Navy, Office of Naval Research, grant #N00014-21-1-2225 and #N00014-22-1-2067, Air Force Office of Scientific Research, grant #22RT0060, and generous gifts from Cisco. The authors would like to thank Austin Vas for this support with the temperature measurements and Simon Thomann for his valuable contributions to the computing-in-memory work.

References

[1] 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.

[2] P. R. Genssler and H. Amrouch, "Brain-inspired computing for circuit reliability characterization," IEEE Transactions on Computers, 2022.

[3] D. Ma, R. Thapa, and X. Jiao, "Molehd: Drug discovery using brain-inspired hyperdimensional computing," arXiv preprint arXiv:2106.02894, 2021.

[4] R. Wang, F. Kong, H. Sudler, and X. Jiao, "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, 2021, pp. 461–464.

[5] R. Wang, X. Jiao, and X. S. Hu, "Odhd: One-class hyperdimensional computing for outlier detection," in *2022 59th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2022.

[6] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," Science Robotics, vol. 4, no. 30, 2019.

[7] A. Mitrokhin, P. Sutor, D. Summers-Stay, C. Fermüller, and Y. Aloimonos, "Symbolic representation and learning with hyperdimensional computing," Frontiers in Robotics and AI, vol. 7, p. 63, 2020.

[8] P. Sutor, D. Yuan, D. Summers-Stay, C. Fermuller, and Y. Aloimonos, *Gluing neural networks symbolically through hyperdimensional computing*, 2022.

[9] P. R. Genssler and H. Amrouch, "Brain-inspired computing for wafer map defect pattern classification," in *IEEE International Test Conference (ITC'21)*, 2021.

[10] P. R. Genssler, A. Vas, and H. Amrouch, "Brain-inspired hyperdimensional computing: How thermal-friendly for edge computing?" IEEE Embedded Systems Letters, 2022.

[11] A. Hernandez-Cane, N. Matsumoto, E. Ping, and M. Imani, "OnlineHD: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 56–61.

[12] M. Hersche, S. Kurella, and T. Schneider, Hyperdimensional Computing Library. 2022.

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

[14] Y.-T. Chen, J. Cong, H. Huang, B. Liu, C. Liu, et al., "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *2012 Design, Automation & Test in Europe Conference (DATE)*, 2012, pp. 45–50.

[15] S. Thomann, H. L. G. Nguyen, P. R. Genssler, and H. Amrouch, "All-in-memory brain-inspired computing using fefet synapses," Frontiers in Electronics, vol. 3, 2022.

[16] S. Thomann, P. R. Genssler, and H. Amrouch, *Hw/sw co-design for reliable in-memory brain-inspired hyperdimensional computing*, 2022.

[17] S. Natarajan, M. Agostinelli, S. Akbar, M. Bost, A. Bowonder, et al., "A 14nm logic technology featuring 2nd-generation finfet, air-gapped interconnects, self-aligned double patterning and a 0.0588 μm2 sram cell size," in *2014 IEEE International Electron Devices Meeting*, 2014, pp. 3.7.1–3.7.3.

[18] S. Zhang, R. Wang, J. J. Zhang, A. Rahimi, and X. Jiao, "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, 2021, pp. 211–217.

[19] S. Zhang, R. Wang, D. Ma, J. J. Zhang, X. Yin, et al., "Energy-efficient brain-inspired hyperdimensional computing using voltage scaling," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 52–55.

[20] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[21] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 739–743.

[22] R. Thapa, D. Ma, and X. Jiao, "Hdxplore: Automated blackbox testing of brain-inspired hyperdimensional computing," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2021, pp. 90–95.

[23] D. Ma, J. Guo, Y. Jiang, and X. Jiao, "Hdtest: Differential fuzz testing of brain-inspired hyperdimensional computing," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2021, pp. 391–396.

[24] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," Advances in neural information processing systems, vol. 30, 2017.

[25] R. Wang and X. Jiao, "Poisonhd: Poison attack on brain-inspired hyperdimensional computing," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 298–303.

[26] P. Poduval, A. Zakeri, F. Imani, H. Alimohamadi, and M. Imani, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," Frontiers in Neuroscience, p. 5, 2022.

[27] B. M. Tijms, A. M. Wink, W. de Haan, W. M. van der Flier, C. J. Stam, et al., "Alzheimer's disease: Connecting findings from graph theoretical studies of brain networks," Neurobiology of aging, vol. 34, no. 8, pp. 2023–2036, 2013.

[28] Z. Zou, Y. Kim, F. Imani, H. Alimohamadi, R. Cammarota, et al., "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.

[29] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, et al., "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, IEEE, 2019, pp. 435–446.

[30] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, IEEE, 2017, pp. 445–456.

[31] A. Mitrokhin, C. Fermüller, C. Parameshwara, and Y. Aloimonos, "Event-based moving object detection and tracking," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1–9.

[32] P. Sutor, D. Summers-Stay, and Y. Aloimonos, "A computational theory for life-long learning of semantics," in *International Conference on Artificial General Intelligence*, Springer, 2018, pp. 217–226.

[33] A. Z. Zhu, D. Thakur, T. Ozaslan, B. Pfrommer, V. Kumar, et al., "The multivehicle stereo event camera dataset: An event camera dataset for 3d perception," IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 2032–2039, 2018.