Adaptive Neural Recovery for Highly Robust Brain-like Representation

Prathyush Poduval[1], Yang Ni[5], Yeseong Kim[2], Kai Ni[3], Raghavan Kumar[4]
Rossario Cammarota[4] and Mohsen Imani[5]*
[1]Indian Institute of Science, [2]Daegu Gyeongbuk Institute of Science and Technology, [3]Rochester Institute of Technology
[4]Intel Labs, [5]University of California Irvine

*Corresponding author: m.imani@uci.com

ABSTRACT

Today's machine learning platforms have major robustness issues dealing with insecure and unreliable memory systems. In conventional data representation, bit flips due to noise or attack can cause value explosion, which leads to incorrect learning prediction. In this paper, we propose RobustHD, a robust and noise-tolerant learning system based on HyperDimensional Computing (HDC), mimicking important brain functionalities. Unlike traditional binary representation, RobustHD exploits a redundant and holographic representation, ensuring all bits have the same impact on the computation. RobustHD also proposes a runtime framework that adaptively identifies and regenerates the faulty dimensions in an unsupervised way. Our solution not only provides security against possible bit-flip attacks but also provides a learning solution with high robustness to noises in the memory. We performed a cross-stacked evaluation from a conventional platform to emerging processing in-memory architecture. Our evaluation shows that under 10% random bit flip attack, RobustHD provides a maximum of 0.53% quality loss, while deep learning solutions are losing over 26.2% accuracy.

ACM Reference Format:
Prathyush Poduval[1], Yang Ni[5], Yeseong Kim[2], Kai Ni[3], Raghavan Kumar[4], Rossario Cammarota[4] and Mohsen Imani[5]*. 2022. Adaptive Neural Recovery for Highly Robust Brain-like Representation. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530659

1 INTRODUCTION

For a long time, storing information in hardware was a solution to ensure the security and reliability of data. Addressing to a particular memory location could be restricted to ensure that non-authorized users cannot access the memory values. However, security and privacy face another level of uncertainty, as values in memory can also be attacked without physically accessing the memory values. In recent years, multiple attacks have been introduced to change the memory values without directly accessing the memory cells, such as the Row Hammer attack [1]. Although researchers are working on solutions to eliminate such memory attacks, there is no evidence showing the memory's inviolability to other unknown attack mechanisms. In other words, memory technologies might be vulnerable to other attacks that have not been discovered yet. Furthermore, from an efficiency point of view, memory technologies, e.g., SRAM, DRAM, and SSD, have been optimized by the industry for maximum density. Changing the memory blocks to ensure security for each new possible attack could be significantly costly and less desirable.

Besides bit-flip attacks, emerging memory devices have various reliability issues such as endurance, durability, and variability [2]. The security and unreliability of memory systems created several challenges in the machine learning area [3, 4]. Regardless of the learning algorithm, hardware platforms store the trained learning model in their memory blocks. In traditional data representation, flipping the exponent or most significant bits can increase the weight value to extremely large, thus changing the prediction result. For example, prior work showed how few bit flips on Deep Neural Network (DNN) model can result in a major change in the prediction result [5]. Unfortunately, existing learning solutions have almost no robustness to possible noise or bit-flip attacks in memory.

In contrast, the human brain can do much of this learning effortlessly and efficiently without much concern about noisy neurons [6]. Every minute, our brain is losing thousands of neurons while providing the same quality of learning. To more closely model the human brain, earlier researchers proposed HyperDimensional Computing (HDC) as an alternative computing method, which mimics important brain functionalities towards self-adaptive, high-efficiency, and noise-tolerant computation [6]. HDC is motivated by the observation that the human brain operates on high-dimensional representations of data. In HDC, objects and data are thereby encoded with high-dimensional vectors, called *hypervectors*, which have 10,000 or more elements [7, 8]. HDC can then perform various learning tasks with computation in the high dimensional space. HDC is well suited to address the efficiency and robustness of learning systems, as: (i) HDC is inherently redundant with strong robustness to noise and failure [9, 10], (ii) HDC models are computationally efficient to train, highly parallel at heart, and amenable to hardware level optimization [9, 11], (iii) it offers an intuitive and human-interpretable model [12], (iv) HDC offers a complete computational paradigm that can be applied to cognitive as well as learning problems [13, 14], and (v) HDC naturally enables low-cost secure and private learning [15].

In this paper, we propose RobustHD, a hyperdimensional self-learning system robust to technology noise and bit-flip attack. Our solution exploits a redundant and holographic representation, ensuring all bits have the same impact on computing. The main contributions of the paper are listed below:

- RobustHD exploits hyperdimensional learning by transforming data into high-dimensional representation. Since encoding spreads the data over a very large hypervector, a substantial number of bits can be corrupted while preserving sufficient information, resulting in the high robustness of HDC to noise and bit-flip attacks.

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9142-9/22/07.
https://doi.org/10.1145/3489517.3530659

- Similar to the human brain, we propose a framework that self-recovers the HDC model to regenerate dimensions with possible attack or noise. Our solution splits the HDC dimensions into smaller chunks. During runtime, RobustHD adaptively identifies possible noisy or attacked chunks and adaptively regenerates them in an unsupervised way.
- We show the advantage of RobustHD to (i) increase the robustness of today's computing systems without any changes on the hardware, (ii) increase the feasibility of processing in-memory platform by enhancing the memory lifetime during computation, and (iii) eliminate the necessity of using costly error correction code or costly refresh cycle in memory blocks.

We performed a cross-stacked evaluation from a conventional platform to emerging processing in-memory architecture. Our evaluation shows that under 10% random (targeted) bit flip, RobustHD provides less than 2.7% (2.9%) quality loss, while deep neural networks are losing over 26.2% (68.1%) of accuracy. Our RobustHD self-learning framework can further reduce the quality loss to 0.53% by adaptively identifying and regenerating the faulty dimensions. In addition, RobustHD systematically enhances the feasibility of processing in-memory platforms by increasing the lifetime from three months to five years.

2 RELATED WORK AND MOTIVATION

Bit-Level Attack & Technology Noise: The adversarial attack has been widely used to evaluate the robustness of machine learning methods. However, there are still several concerns about the effect of model parameters/weights on learning accuracy. The deep neural networks (DNNs) parameters have been attacked using different hardware trojans, which require a specific pattern of input to trigger the trojan inside the network [16]. In many scenarios, such trojan attacks may not be feasible as the attacker does not have access to make the hardware-level modification. Fault injection attacks are another type of attack on machine learning parameters. For example, a single bias attack changes a certain bias term in a neuron to change the classification result. Prior work also injected fault into the activation function of DNN to misclassify a target input [5].

Several recent works studied the possibility of a bit-flip attack in memory technologies, mainly targeting machine learning applications. An example of this attack is a Row Hammer attack [3, 4] that attempts to change learning parameters stored in DRAM. These methods are focused on flipping extremely vulnerable bits, e.g., exponent bits in floating-point precision. The explanation behind that is flipping the exponent bit can increase the weight value to extremely large, thus leading to exploding the prediction result. From the other side, recent DNN accelerators, e.g., Google's Tensor Processing Units (TPU) [17], are working with quantized 8-bit values. Although this representation gives higher robustness to parameter perturbation, work in [5] introduced a method to maliciously cause a DNN system malfunction by flipping an extremely small amount of vulnerable bits in the network weights. The technological and fabrication issues in highly scaled technology are another source of noise that can significantly change the machine learning behavior [2, 18].

Hyperdimensional computing (HDC): The brain's circuits are massive in terms of numbers of neurons and synapses, suggesting that large circuits are fundamental to the brain's computing. Hyperdimensional computing (HDC) [6] explores this idea by looking at computing with ultra-wide words – that is, with very high-dimensional vectors, or hypervectors. This lets us combine such hypervectors into a new hypervector using well-defined vector space operations while keeping the information of the two with high probability. Hypervectors are holographic and (pseudo)random with i.i.d. components. A hypervector contains all the information combined and spread across all its components in a full holistic representation so that no element is more responsible for storing any piece of information than another.

In this paper, for the first time, we explore the effectiveness of hyperdimensional representation to provide inherent robustness against noise and attack to machine learning parameters. We also propose a novel self-recovery framework that adaptively identifies and regenerates faulty bits of the HDC learning model at runtime in an unsupervised way. Our solution can perform an accurate learning prediction even under serious unknown bit-flip attacks or possible hardware failure.

Challenges in Binary Data Representation: In existing machine learning algorithms, e.g., DNNs, weights represent using fixed-point or floating-point value. A single error in the hardware, e.g., bit flip in the memory store DNN weights, can result in different levels of errors on the weight value. We evaluate the robustness of the DNN model on multiple datasets (listed in Table 2) using an 8-bit fixed-point representation. Our evaluation shows that a 2% random (targeted) bit flip attack can cause 7.9% (13.5%) quality loss in DNN.

3 HYPERDIMENSIONAL LEARNING

3.1 HDC Learning Steps

The first step in HDC is to map each data points into high-dimensional space. The mapping procedure is often referred to as *encoding*. Assume an input vector (an image, voice, etc.) in original space $\vec{F} = \{f_1,\ f_2, \cdots, f_n\}$ and $F \in \mathcal{R}^n$. The encoding module maps this vector into high-dimensional vector, $H \in \{0,1\}^D$, where $D >> n$. The following equation shows an encoding method that maps input vector into high-dimensional space [19]: $\vec{\mathcal{H}} = \sum_{k=1}^{n} |f_k|_{\in \mathcal{F}} \oplus \vec{\mathcal{B}}_k$, where $\oplus$ denotes binding (XOR operation) and $\vec{\mathcal{B}}_k$s are randomly chosen base hypervectors of dimension $\mathcal{D} \simeq 10k$ to retain the spatial or temporal location of features in an input. That is, $\vec{\mathcal{B}}_k \in \{0,1\}^{\mathcal{D}}$ and $\delta(\vec{\mathcal{B}}_{k_1}, \vec{\mathcal{B}}_{k_2}) \simeq D/2$, where δ denotes Hamming distance similarity.

To find the universal property for each class in the training dataset, the trainer module linearly combines hypervectors belonging to each class, i.e., adding the hypervectors to create a single hypervector for each class. Once combining all hypervectors, we treat per-class accumulated hypervectors, called *class hypervectors*, as the learned model. Assuming a problem with k classes, the model represents using: $\mathcal{M} = \{\vec{C}_1, \vec{C}_2, \cdots, \vec{C}_k\}$. For example, after generating all encoding hypervector of inputs belonging to class/label l, the class hypervector $\vec{C}^l$ can be obtained by bundling (adding) all $\vec{\mathcal{H}}^l$s. Assuming there are $\mathcal{J}$ inputs having label l: $\vec{C}^l = \sum_j^{\mathcal{J}} \vec{\mathcal{H}}_j^l$

HDC checks the Hamming distance similarity of each encoded test data with the class hypervector in two steps. The first step encodes the input (the same encoding used for training) to produce a query hypervector $\vec{\mathcal{H}}$. We compute the similarity (δ) of $\vec{\mathcal{H}}$ and all class hypervectors. Query data gets the label of the class with the highest Hamming similarity.

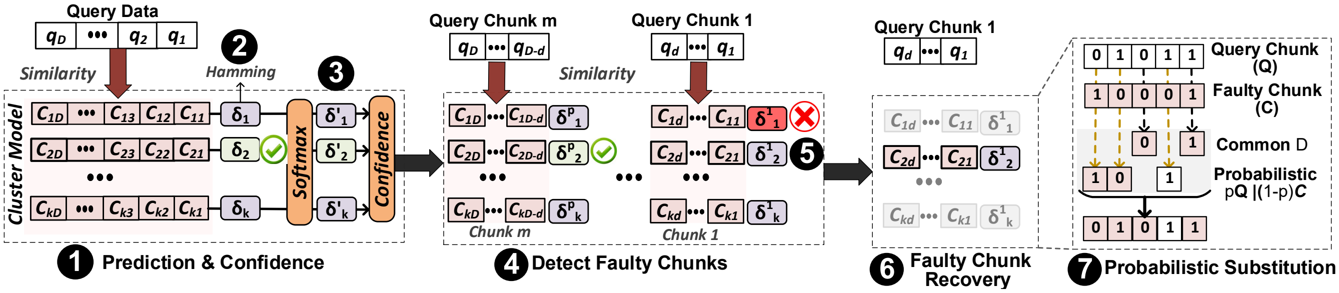
Query Data
q_D ••• q_2 q_1
Similarity
❷ Hamming
❸
Cluster Model
C_{1D} ••• C_{13} C_{12} C_{11} δ_1
C_{2D} ••• C_{23} C_{22} C_{21} δ_2 ✓
C_{kD} ••• C_{k3} C_{k2} C_{k1} δ_k
Softmax
δ'_1 δ'_2 δ'_k
Confidence
❶ Prediction & Confidence
Query Chunk m
q_D ••• q_{D-d}
Query Chunk 1
q_d ••• q_1
Similarity
C_{1D} ••• C_{1D-d} δ^p_1
C_{2D} ••• C_{2D-d} δ^p_2 ✓
C_{kD} ••• C_{kD-d} δ^p_k
Chunk m
C_{1d} ••• C_{11} δ^1_1 ✗
C_{2d} ••• C_{21} δ^1_2 ❺
C_{kd} ••• C_{k1} δ^1_k
Chunk 1
❹ Detect Faulty Chunks
Query Chunk 1
q_d ••• q_1
C_{1d} ••• C_{11} δ^1_1
C_{2d} ••• C_{21} δ^1_2
C_{kd} ••• C_{k1} δ^1_k
❻ Faulty Chunk Recovery
0 1 0 1 1 Query Chunk (Q)
1 0 0 0 1 Faulty Chunk (C)
0 1 Common D
1 0 1 Probabilistic pQ |(1-p)C
0 1 0 1 1
❼ Probabilistic Substitution

Figure 1: Overview of RobustHD framework identifying and data recovery from faulty dimensions in hyperdimensional system.

Table 1: HDC quality loss under random noise using models with different precision and dimensionality.

Hardware Error		*1%*	*2%*	*5%*	*10%*	*15%*
DNN		3.9%	9.4%	16.3%	26.4%	40.0%
D=5k	**1-bit**	0.0%	0.0%	0.9%	3.1%	5.2%
	2-bits	0.0%	0.4%	1.4%	4.7%	7.9%
D=10k	**1-bit**	0.0%	0.0%	0.6%	1.7%	3.3%
	2-bits	0.0%	0.2%	1.1%	3.5%	5.7%

3.2 Hypervector Representation

One of the main advantages of HDC is its high robustness to noise and failure. In HDC, 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. HDC efficiency and robustness depend on two parameters: (i) the hypervector dimensionality that determines the hypervector capacity and the level of redundancy, and (ii) the precision of each hypervector element. Increasing dimensionality or precision of elements results in improving the classification accuracy. However, increasing dimensionality results in an efficiency issue, while high precision representation reduces the computation robustness.

Table 1 show the computational robustness of HDC to a different percentage of random noise in hardware. The results are reported for a human activity recognition task (UCIHAR) [20] using hypervectors with different dimensions ($D = 5k$ and $D = 10k$) and different bit precision. Our results indicate that HDC provides higher computational robustness when using a high-dimensional and lower precision model. To ensure robustness, we always use HDC with a binary model. This makes HDC ideal for a robust and secure learning method, as even a substantial noise cannot change the prediction result.

4 ROBUSTHD SELF DATA RECOVERY

In this section, we propose RobustHD, a framework that identifies noisy or attacked dimensions of the hyperdimensional learning model stored in the memory. Next, RobustHD regenerates the faulty model dimensions in order to overcome the noise accumulation that can cause the wrong HDC prediction. The main challenge in RobustHD is detecting the noisy elements, and their recovery needs to happen in an unsupervised way, without accessing any training data. This is a challenging process since the entire memory is vulnerable to attack. This makes it impossible to assume there is any non-faulty copy of the HDC model stored in memory. Figure 1 shows an overview of RobustHD framework. RobustHD exploits inference (unlabeled) data in order to identify and recover the faulty dimensions in the HDC model. For each inference data, the HDC model makes a prediction. For queries with high confidence, we consider the prediction result as a possible label for the query. Then, RobustHD splits the HDC dimensions into smaller chunks and identifies chunks that negatively affect the prediction as potential faulty dimensions. Finally, RobustHD updates the noisy chunks by inherently substituting the chunk elements with query bits.

4.1 RobustHD Prediction Confidence

As explained in Section 3.1, during the inference, RobustHD checks the similarity of a query with all class hypervectors (❶). A query data assigns to a class with the highest similarity. Since class hypervectors and queries are binary, we use Hamming distance as a similarity metric (❷). RobustHD defines confidence for each prediction bypassing the similarity values (δ) through a normalization block, i.e., softmax. The $\delta'(\vec{Q}, \vec{C}_i)$ indicates the confidence of each class hypervector (❸). The confidence not only determines how similar a query is with a certain class but also what its margin is to other class hypervectors. For query data with higher confidence than a threshold (T_C), RobustHD trusts the prediction results as a possible label for the query.

4.2 Noisy Chunk Detection

Let us consider a query $\vec{Q}$ matched with high confidence with class $\vec{C}^2$, in an example shown in Figure 1(❹). RobustHD splits the query and class hypervectors into m smaller chunks with a size of d, where $d = D/m$. Next, RobustHD looks at each chunk as a separate HDC model and checks the correctness of the corresponding query (d dimensional query) with our expected class, $\vec{C}^2$. We consider all chunks with a correct prediction as healthy chunks, while mismatched chunks are considered faulty (❺).

4.3 Probabilistic Substitution

Assume $\vec{A}$ and $\vec{B}$ are two randomly generated vectors. In order to bring vector $\vec{A}$ closer to vector $\vec{B}$, a random (typically small) subset of vector $\vec{B}$'s indices is forced onto vector $\vec{A}$ by setting those indices in vector $\vec{A}$ to match the bits in vector $\vec{B}$. Therefore, the Hamming distance between vector $\vec{A}$ and $\vec{B}$ is made smaller through partial cloning. When vector $\vec{A}$ and $\vec{B}$ are already similar, then indices selected probably contain the same bits, and thus the information in $\vec{A}$ does not change (❻). The advantage of this operation is that it involves no arithmetic, leading to fast and memory-efficient hardware implementation. Binary substitution updates each dimension of the selected class stochastically with p probability: $p\vec{Q}|(1-p)\vec{C}$.

In other words, with flip probability p, each element of the selected class hypervector will be replaced with the elements of the query hypervector (❼). Using a small probability is conservative, as the model will have minor changes during data recovery. A larger learning rate will result in major changes in a model after each iteration, resulting in a higher probability of divergence.

5 PROCESSING IN-MEMORY

Running big data algorithms with large datasets on conventional processors results in high energy consumption and slow processing speed. Although new processor technology has evolved to serve computationally complex tasks more efficiently, data movement costs between the processor and memory still hinder the higher efficiency of application performance. Processing in-memory (PIM) is a promising solution to accelerate applications with a large amount of parallelism. Several recent works have explored the advantage of SRAM, DRAM, and non-volatile memory to enable PIM functionality to accelerate big data applications [21, 22, 23]. In this work, we focus on developing a digital-based PIM architecture (called RobustHD) that exploits the switching characteristic of Non-Volatile Memory (NVM) technologies to perform bitwise operations in memory [23, 21, 22]. DPIM is a tensor-based processor supporting an extensive amount of vector operation locally in memory.

In this paper, we exploit DPIM arithmetic to accelerate existing machine learning solutions. Then, we explain major robustness challenges that makes today's DPIM platforms infeasible to operate over existing data representation. Finally, we introduce hyperdimensional computing as an ideal representation for DPIM and other noisy PIM platforms.

5.1 PIM Supported Operations

PIM supports arithmetic operations directly on digital data stored in memory without reading them out of sense amplifiers [23, 24]. Our design exploits the NVM switching characteristic to implement NOR gates in digital memory [24, 23]. Our solution selects three or more columns of the memory as input NOR operands by connecting them to ground voltage. Next, DPIM connects the bitline corresponding to the output of NOR operation to a write voltage (V_0). In addition, all outputs NVMs located in the output column are initialized to R_{ON} in the beginning. To execute NOR in a row, an execution voltage, V_0, is applied at the p terminals of the inputs while the p terminal of the output NVM is grounded, as shown in Figure **??**. During NOR computation, the output memristor is switched from R_{ON} (blue color) to R_{OFF} (red color) when one or more inputs stored '1.' value (R_{ON}). In fact, the low resistance input passes a current through an output NVM resulting in writing R_{off} value on it. This NOR computation performs in row-parallel on all the activated memory rows by the row-driver.

5.2 Endurance & Memory

NVM devices have limited endurance; thus, their lifetime depends on the number of write operations. During memory functionality, NVM devices often do not face many write operations. However, the memory can be still become unreliable after many write operations on NVM cells. To overcome the endurance challenge and increase the lifetime, memory technologies often equipped with wear-leveling and error correction codes (ECC) techniques. The wear-leveling uniformly distributes the write operations over all memory blocks to ensure the endurance similarly affects all cells. Memory technologies also use ECC to detect and recover possible errors in memory. The usage of ECC becomes more important when the memory blocks are more impacted by endurance. The cost of ECC can dominate the system performance when we deal with noisy memory blocks.

Table 2: Datasets (*n*: feature size, *k*: number of classes)

	n	k	**Train Size**	**Test Size**	**Description**
MNIST	784	10	60,000	10,000	Handwritten Recognition[25]
UCIHAR	561	12	6,213	1,554	Activity Recognition(Mobile)[20]
ISOLET	617	26	6,238	1,559	Voice Recognition [26]
FACE	608	2	522,441	2,494	Face Recognition[27]
PAMAP	75	5	611,142	101,582	Activity Recognition(IMU) [28]
PECAN	312	3	22,290	5,574	Urban Electricity Prediction [29]

5.3 Endurance & Processing In-Memory

Unlike the memory functionality, the PIM operations are causing extensive switching and write operations in NVM devices. This can significantly affect the reliability of the devices and the functionality of an accelerator as a whole. The PIM architecture is going to have a more negative impact on endurance when performing high precision arithmetics. In the PIM platform, the number of sequential cycles (the number of write operations) is increasing quadratically with the bit-width during PIM multiplication operation. On another side, using high precision values increases the possibility of a bit flip in MSBs or exponents positions that can cause major changes in the value. Unlike existing machine learning algorithms that work over traditional data representation, our HDC solution operates over high-dimensional vectors with binary representation. This makes HDC significantly robust to operate over unreliable devices. In Section 6.5, we show PIM lifetime accuracy accelerating DNN and HDC algorithm.

6 EVALUATION

6.1 Experimental Setup

We implement RobustHD using both software and hardware support. In software, we verified RobustHD functionality using Python implementation. In hardware, we designed a cycle-accurate simulator based on TensorFlow [30] that emulates memory functionality running the neural network and hyperdimensional computing. For the hardware design, we use HSPICE for circuit-level simulations to measure the energy consumption and performance of all the PIM operations in 28nm technology. PIM works with any bipolar resistive technology, which is the most commonly used in existing NVMs. In order to have the highest similarity to commercially available 3D Xpoint, we adopt the memristor device with a VTEAM model [31]. The memristor's model parameters are chosen to produce a switching delay of 1ns, a voltage pulse of 1V and 2V for RESET and SET operations to fit practical devices [32].

We evaluate RobustHD accuracy and efficiency on six popular datasets (listed in Table 2), ranging from small data collected in a small IoT network to large data, which includes hundreds of thousands of images of facial data. We compare RobustHD accuracy and efficiency with state-of-the-art deep neural network solutions. All network configurations are selected based on the work in [33].

6.2 RobustHD Accuracy

We compare RobustHD accuracy with state-of-the-art learning methods when assuming a different level of faulty bits. In all implementations, we assume the trained model, i.e., model weight,

Table 3: HDC quality loss using different number of bits.

Error Rate		2%	4%	6%	8%	10%	12%
DNN	*Random*	7.9%	8.4%	16.6%	21.0%	26.2%	29.6%
	Targeted	13.5%	15.9%	34.8%	50.5%	68.1%	80.0%
SVM	*Random*	3.7%	5.3%	8.9%	13.4%	16.1%	22.4%
	Targeted	5.6%	9.0%	16.9%	28.1%	35.9%	53.1%
AdaBoost	*Random*	1.3%	2.5%	2.9%	4.2%	7.3%	11.6%
	Targeted	3.4%	6.5%	7.5%	10.9%	19.0%	30.2%
HDC	*Random*	0.7%	1.0%	1.6%	2.0%	2.7%	3.2%
	Targeted	0.7%	1.1%	1.8%	2.3%	3.1%	3.3%

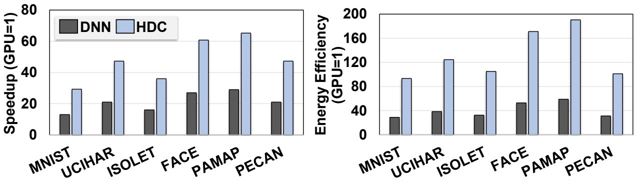

Figure 2: PIM efficiency running DNN and HDC.

are stored in a memory that is possibly vulnerable to attack or error. Table 3 compares RobustHD classification accuracy with deep neural network and boosting algorithms under different levels of faulty bits. The results are reported for two scenarios: *(1) Random Attack:* where any arbitrary bits can be flipped, and *(2) Targeted Attack:* where the error is the worst-case, and it occurs on the most significant bits. Our evaluation shows that existing learning solutions have a very high sensitivity to faulty bits. This sensitivity is significantly higher using targeted attack since each bit-flip makes greater changes in the weight values. For example, DNN and AdaBoost lose 29.6% and 11.6% (80.0% and 30.2%) assuming 12% random (targeted) bit flip on the memory storing the learning weights. In contrast, RobustHD has significantly high robustness to faulty bits. RobustHD has holographic binary distribution where all bits have the same impact on the computation. This, along with the high-dimensionality (redundancy) of HDC representation, makes HDC significantly robust against possible error rate in the memory. In addition, RobustHD has the same behavior using a random or targeted bit attack. For example, HDC provides 3.2% and 3.3% quality loss using a 12% random and targeted bit-flip attack.

6.3 RobustHD Efficiency

Figure 2 compares RobustHD efficiency with the state-of-the-art learning solutions running on processing in-memory architecture. All results are normalized to DNN running on NVIDIA 1080 GTX GPU. Both PIM and GPU platforms are working based on the TensorFlow backend, aiming to maximize the throughput. Our evaluation shows that our PIM platform significantly accelerates DNN and HDC computation as compared to GPU architecture. This efficiency comes from PIM's capability to address the data movement issue as well as providing extensive parallelism. Our results also indicate that HDC provides higher computation efficiency as compared to DNN running on the PIM platform. This higher efficiency comes from (i) the parallelism supported by the HDC algorithm, which translates the learning task to simple and parallel operations over hypervectors, (ii) simplifying the learning process to bitwise operations, rather than DNNs that use non-friendly fixed-point operations. Our evaluation shows that HDC provides 2.4× and 3.7× (47.6× and 21.2×) faster and higher energy efficiency compared to DNN running on PIM (GPU).

Table 4: Quality loss with/without RobustHD data recovery.

Error Rate		MNIST	UCIHAR	ISOLET	FACE	PAMAP	PECAN
Without Recovery	*2%*	0.46%	0.93%	0.14%	0.32%	0.68%	1.61%
	6%	1.77%	1.96%	0.79%	1.43%	1.80%	2.14%
	10%	2.75%	3.18%	1.30%	2.47%	2.94%	3.7%
With Recovery	*2%*	0%	0%	0%	0%	0%	0%
	6%	0.10%	0.17%	0.07%	0.19%	0.15%	0.16%
	10%	0.26%	0.48%	0.44%	0.28%	0.42%	0.53%

6.4 RobustHD Data Recovery

We show the impact of RobustHD self data recovery on the accuracy of HDC learning systems. In existing learning systems, we cannot identify and recover a faulty bit since we assume there is no secure memory that can store the non-faulty version of model weights. In contrast, RobustHD inherently supports a self data recovery method that can identify and fix possible noisy bits in an unsupervised way. Table 4 reports the classification accuracy of HDC classification accuracy with and without RobustHD framework assuming different faulty bits. Our evaluation shows that HDC without RobustHD starts losing accuracy when the portion of bit flip increases. For example, with 6% (12%) bit flip error, RobustHD reducing the quality loss to 0.15% (0.39%).

Confidence Level: The confidence value (T_C) and the rate of bit substitution (S) significantly impact RobustHD effectiveness on faulty bit recovery. Figure 3 shows the number of samples required by RobustHD to recover noisy dimensions and the final quality loss after RobustHD data recovery. Using a large T_C reduces the number of data samples that can update the faulty dimensions. This results in a higher rate of error accumulation on the HDC model (when the rate of bit flip is high), which eventually results in a high-quality loss. In addition, a larger T_C only trusts high confident data to update the model, thus enabling less fluctuation on the accuracy after each data recovery. In contrast, a low T_C updates the HDC model more frequently but with a more destructive update process. This causes fluctuation in the model accuracy during RobustHD data recovery.

Substitution Rate: Figure 3 also shows the impact of bit substitution, S, on the effectiveness of RobustHD framework. Our evaluation shows that the higher S makes the HDC model more robust against possible bit-flip attacks. However, this can cause lower classification accuracy as HDC cannot rely on the dimensions of a single query to update a chunk. Similarly, the low substitution rate results in a lower data recovery rate, leading to error accumulation on class hypervectors (when the substitution rate is lower than the attack rate).

6.5 RobustHD & In-Memory Lifetime

Figure 4a shows the classification accuracy of PIM architecture accelerating DNN and HDC algorithms. The results are reported for the NVM device with 10^9 endurance [2]. As explained in Section 5, PIM arithmetic operations involve several write operations in memory, which can cause device failure due to limited endurance in NVM devices. In conventional data representation, this failure can result in significant quality loss in learning accuracy. In contrast, HDC has high robustness to device failure. As Figure 4a shows, PIM running DNNs will start losing accuracy in less than three months. A higher quality loss is observed when using high-precision or floating-point precision. This is because bit failures on high-precision values can potentially cause larger changes in the computation, e.g., weight values in DNN. In contrast, HDC

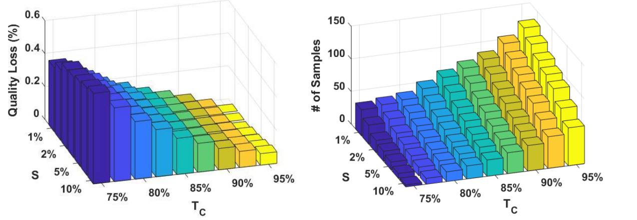

Figure 3: Impact of confidence & substitution on data recovery.

provides very robust accuracy results over the years of using the PIM processor. RobustHD robustness is increasing with the size of hypervector dimensions. RobustHD with $D = 4k$ and $D = 10k$ can provide the same learning accuracy (less than 1% quality loss) during 3.4 years and 5 years, respectively. Our solution makes the PIM platform more feasible to be used as an accelerator.

6.6 DRAM Relaxation for Higher Efficiency

In DRAM, most of the power is consumed to perform refresh cycles. In each cycle, DRAM constantly (every 64 milliseconds) overwrite decaying bits. This process consumes a major portion of DRAM power consumption. Recently, multiple industries have been aiming to design low-power DRAM in which the refresh rate of the DRAM was reduced in order to cut power consumption. However, reducing the refresh time causes errors in the memory block. As we explained in Section 2, in traditional data representation, a bit-flip error can cause significant changes in the memory value. In contrast, in our HDC representation, we have higher redundancy and robustness to noise in each value. We observe that one of the main advantages of RobustHD is its capability to eliminate expensive error correction codes. RobustHD framework inherently recovers the faulty bits stored in memory while improving the total memory efficiency by removing error correction costs. Figure 4b compares the efficiency improvement of DRAM storing models of DNN and HDC. DRAM with a conventional 64ms refresh cycle has almost no error rate; thus, both DNN and HDC models can provide maximum learning accuracy. Relaxing the memory refresh time can cause memory to have bit errors while making the memory more energy efficient. Our evaluation shows that relaxing DRAM refreshing time to have a 4% (6%) error rate can result in 14% (22%) improvement in DRAM energy efficiency. As we explained in Section 6.2, these error rates have a low impact on RobustHD quality loss.

7 CONCLUSION

In this paper, we propose RobustHD, a robust and noise-tolerant learning system based on hyperdimensional computing, mimicking important brain functionalities. RobustHD exploits a redundant and holographic representation, ensuring all bits have the same impact on the computation. RobustHD also presents a framework that adaptively identifies and regenerates the faulty dimensions in an unsupervised way at runtime. Our solution not only provides security against possible bit-flip attack but also provides a learning solution with high robustness to noises in the memory technology.

ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation #2127780, Semiconductor Research Corporation (SRC) Task #2988.001, Department of the Navy, Office of Naval Research, grant #N00014-21-1-2225, Air Force Office of Scientific Research, and a generous gift from Cisco.

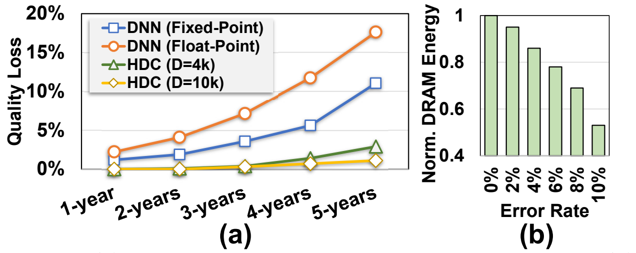

Figure 4: (a) Memory lifetime during PIM functionality, (b) Impact of DRAM refresh cycle relaxation on efficiency.

REFERENCES

[1] L. Cojocar *et al.*, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *SP*, pp. 55–71, IEEE, 2019.

[2] J. B. Kotra *et al.*, "Re-nuca: A practical nuca architecture for reram based last-level caches," in *IEEE IPDPS*, pp. 576–585, IEEE, 2016.

[3] Y. Liu *et al.*, "Fault injection attack on deep neural network," in *ICCAD*, pp. 131–138, IEEE, 2017.

[4] J. Breier *et al.*, "Practical fault attack on deep neural networks," in *ACM SIGSAC CCS*, pp. 2204–2206, 2018.

[5] A. S. Rakin *et al.*, "Bit-flip attack: Crushing neural network with progressive bit search," in *ICCV*, pp. 1211–1220, 2019.

[6] 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.

[7] Z. Zou *et al.*, "Spiking hyperdimensional network: Neuromorphic models integrated with memory-inspired framework," *arXiv preprint arXiv:2110.00214*, 2021.

[8] A. Hernández-Cano *et al.*, "Reghd: Robust and efficient regression in hyper-dimensional learning system," in *DAC*, pp. 7–12, IEEE, 2021.

[9] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.

[10] A. Hernandez-Cane *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *DATE*, pp. 56–61, IEEE, 2021.

[11] M. Imani *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *HPCA*, pp. 221–234, IEEE, 2021.

[12] P. Poduval *et al.*, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, 2022.

[13] A. Mitrokhin *et al.*, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, 2019.

[14] R. Chen *et al.*, "Joint active search and neuromorphic computing for efficient data exploitation and monitoring in additive manufacturing," *Journal of Manufacturing Processes*, vol. 71, pp. 743–752, 2021.

[15] A. Hérnandez-Cano *et al.*, "Prid: Model inversion privacy attacks in hyperdimensional learning systems," in *DAC*, pp. 553–558, IEEE, 2021.

[16] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," *arXiv preprint arXiv:1806.05768*, 2018.

[17] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, pp. 1–12, IEEE, 2017.

[18] H. Saadeldeen *et al.*, "Memristors for neural branch prediction: a case study in strict latency and write endurance challenges," in *ACM CF*, pp. 1–10, 2013.

[19] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *CLOUD*, pp. 435–446, IEEE, 2019.

[20] D. Anguita *et al.*, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *IWAAL*, Springer, 2012.

[21] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *MICRO*, pp. 273–287, IEEE, 2017.

[22] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *ISCA*, pp. 383–396, IEEE, 2018.

[23] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ACM/IEEE ISCA*, pp. 802–815, IEEE, 2019.

[24] S. Kvatinsky *et al.*, "Magic—memristor-aided logic," *TCAS II*, vol. 61, no. 11, 2014.

[25] D. Ciregan *et al.*, "Multi-column deep neural networks for image classification," in *CVPR*, pp. 3642–3649, IEEE, 2012.

[26] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.

[27] A. Angelova *et al.*, "Pruning training sets for learning of object categories," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005*, IEEE, 2005.

[28] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *ISWC*, pp. 108–109, IEEE, 2012.

[29] "Pecan street dataport." https://dataport.cloud/.

[30] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[31] S. Kvatinsky *et al.*, "Vteam: A general model for voltage-controlled memristors," *IEEE TCAS II*, vol. 62, no. 8, pp. 786–790, 2015.

[32] R. Ben-Hur *et al.*, "Simpler magic: Synthesis and mapping of in-memory logic executed in a single row to improve throughput," *TCAS I*, vol. 39, no. 10, 2019.

[33] M. S. Razlighi *et al.*, "Looknn: Neural network with no multiplication," in *DATE*, IEEE, 2017.