DeepER-HD: An Error Resilient HyperDimensional Computing Framework with DNN Front-End for Feature Selection

Mohamed Mejri, Chandramouli Amarnath and Abhijit Chatterjee
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332–0250

2024 IEEE 25th Latin American Test Symposium (LATS) | 979-8-3503-6555-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/LATS62223.2024.10534617

***Abstract*—Brain-inspired hyperdimensional (HD) computing models mimic cognition through combinatorial bindings of biological neuronal data represented by high-dimensional vectors and related operations. However, the efficacy of HD computing depends strongly on input signal and data features used to realize such bindings. In this paper, we propose a new HD-computing framework based on a co-trainable DNN-based feature extractor pre-processor and a hyperdimensional computing system. When trained with restrictions on the ranges of hypervector elements for resilience to memory access errors, the framework achieves up to 135% accuracy improvement over baseline HD-computing for error-free operation and up to three orders of magnitude improvement in error resilience compared to the state-of-the-art. Results for a range of applications from image classification, face recognition, human activity recognition and medical diagnosis are presented and demonstrate the viability of the proposed ideas.**

***Index Terms*—Brain-inspired computing, Hyperdimensional (HD) computing, Deep Learning, Error Resilience**

I. Introduction

Hyperdimensional computing (HDC) is a novel computing paradigm inspired by high-dimensional data representations called hypervectors and arithmetic operations on hypervectors designed to mimic the cognitive capabilities of the human brain [1]. It is promising in that it generally offers a smaller model size and reduced computation cost compared to modern deep neural networks. In signal processing applications however, the performance of hyperdimensional computing is highly dependent on the signal features used to build the high-dimensional hypervector representations of data [2]. Such feature selection, if performed empirically, makes HDC computing highly dependent on the *quality* of selected features. HDC is a low power computing paradigm making it suitable for edge devices. However, aggressive technology scaling along with voltage scaling, makes HDC systems vulnerable to soft and timing errors especially in associative memory [3]. One way to overcome such errors is to build in redundancy (e.g, TMR) [3]) into memory systems. However, this incurs high design costs [3]. In relation to the above, the *key contributions of this research* are as follows:

- A system called *DeepER-HD*, an HDC structure with a co-trained deep learning front-end is developed. The express task of the deep learning front-end (CNN/DNN) is to perform signal feature extraction. This front-end is *trained concurrently with the HDC system* and permits up to 69% accuracy improvement over baseline HD-computing.
- DeepER-HD is designed to be *resilient to errors in associative memory* of HDC systems. Resilience is achieved by imposing *range restrictions on the elements of hypervectors* during training of the HDC system. The range restriction technique achieves up to 77.1% improvement in accuracy under high memory access error rates.

In the following, we first discuss prior work related to this research followed by an overview of HD computing. The DeepER-HD architecture is then described. We next discuss how the system is trained for operation with improved feature selection and error resilience. This is followed by experimental simulation results followed by validation through experiments on FPGA hardware.

II. Prior Work

Prior work in hyperdimensional computing [2], [4], [5] has focused primarily on algorithms geared towards high-accuracy and energy-efficient systems for diverse application domains such as voice [6] and face [7] recognition as well as robotics [8]. To reduce the complexity of HDC systems, quantized and binary HDC designs have been proposed [9], [10]. Backpropagation in hyperdimensional learning systems was investigated in [11] and the use of a neural network as an encoder for HDC was proposed in [12], [13]. However, HDC was used to post-process the neural network output with both systems being trained independently of each other. Architectures for image classification [14] achieve high accuracy through use of random Fourier features.

Prior work in error resilience of DNNs has focused on the robustness of deep learning algorithms to soft errors and permanent faults in AI hardware (accelerators, GPUs). To the best of our knowledge, the work of [3] was the first to investigate the robustness of HDC systems to soft and timing errors. An error resilient HDC technique based on adaptive scaling and clipping was introduced by [15]. Algorithms such as [16], [17] and [18] were developed for error resilience in deep neural networks, and we leverage this prior work in our error resilience formulation.

In term of error resilience, we propose a trainable range restriction based framework that provides increased error re-

silience as compared to scaling and clipping. Our DeepER-HD technique offers a balance between accuracy and latency compared to the state-of-the-art.

III. Hyperdimensional Computing: Preliminaries

A standard HDC system consists of a dataset-based encoder which is often represented by a matrix multiplication that transposes the features of the dataset into a hyperdimensional space. In the HDC training step (on the training dataset), the hypervectors that belong to the same class are summed to generate the class hypervector C, which is stored in associative memory. The model parameters are then updated as follows: if the training datapoint x_T is misclassified (i.e the predicted label $\hat{k} \neq k$), the class hypervector $C_{\hat{k}}$ and C_k are updated according to the following equations [19]:

$$C_{\hat{k}} \leftarrow C_{\hat{k}} - \eta \mathcal{E}\left(x_T\right)$$
$$C_k \leftarrow C_k + \eta \mathcal{E}\left(x_T\right)$$

where η is the learning rate, $\mathcal{E}$ is the hypervector encoded training dataset.

The testing phase consists of encoding the test data point to generate a query hypervector h_Q. In Equation 2, the predicted label $\hat{k}$ is the most similar (measured by the cosine function) class hypervector to h_Q. The cosine similarity is measured between h_Q and each of the class hypervector $\mathcal{H}_k$ as in Equation 1, where $\langle . \rangle$ denotes the inner product operation and $\|.\|$ the L_2 norm.

$$cos\left(h_Q, \mathcal{H}_k\right) = \frac{\langle h_Q, \mathcal{H}_k \rangle}{\|h_Q\| \|\mathcal{H}_k\|} \tag{1}$$

$$\hat{k} = argmax_k \left(cos\left(h_Q, \mathcal{H}_k\right)\right) \tag{2}$$

IV. DeepER-HD Architecture and Training

The DeepER-HD architecture contains a feature extractor followed by a hyperdimensional computing system. The feature extractor is discussed next.

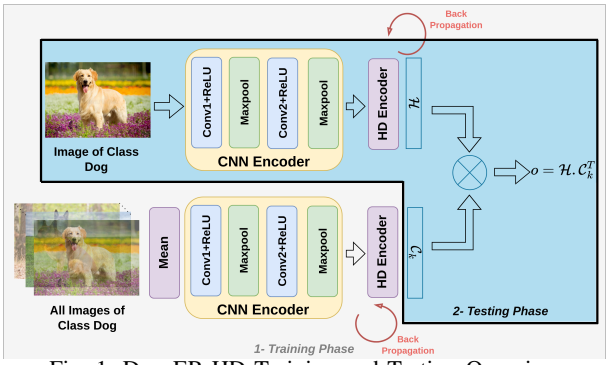


Fig. 1: DeepER-HD Training and Testing Overview

Figure 1 gives an overview of the DeepER-HD system, consisting of feature extractor (CNN Encoder) followed by HDC system. The feature extractor consists of a set of convolutional layers followed by maxpooling layers. Its output is forwarded to a trainable HD encoder (i.e, a trainable linear layer with high dimensional output) Note that in general, the feature extractor can also be based on existing DNN backbones (e.g. ResNet) or use less complex operations (e.g. a mix of 1D convolutional filters and dense layers). During training, for each training image the DeepER-HD framework takes as input ((I) in Fig. 1), it also takes in the average of all images in the training dataset with the same label as (I) ((II) in Fig. 1). The network is optimized to make the training image (I) as similar to the average image. During inference, the query image is fed to the feature extractor and HD encoder, whose output ($\mathcal{H}$) is compared to each class hypervector ($\mathcal{C}_k$) as shown in the figure1. The maximally similar class hypervector determines the predicted class.

A. DeepER-HD Training:

A major drawback of standard HDC training is that the encoding system is frozen while the HD system's model parameters are updated. This is addressed in DeepER-HD by a novel system using iterative optimization techniques (such as gradient descent) to back-propagate the cross entropy loss between the true labels and the predicted probabilities through the feature extractor weights. Algorithm 1 describes the DeepER-HD training procedure. In this work the preprocessor system's feature extractor module is assumed to be a CNN $\mathcal{M}_{cnn}$ with weights denoted by $\mathcal{W}_{cnn}$ and its dense layer has weights denoted by $\mathcal{W}_d$ ($\mathcal{W}_{cnn}$ and $\mathcal{W}_d$ are initially set to a standard normal distribution $\mathcal{N}\left(0, 1\right)$). In line 2 of Algorithm 1, the system is trained for N_{epochs} epochs with a learning rate of η. During each epoch, in line 3, each training datapoint X goes through $\mathcal{M}_{cnn}$ to generate an encoded feature map. This feature map is multiplied by $\mathcal{M}_d$ to generate a training hypervector $\mathcal{H}$. In line 4, the same CNN (feature extractor) module $\mathcal{M}_{cnn}$ followed by the dense layer generates the class hypervectors $\mathcal{C}_H$ from the class vector C. The class vector is composed of K elements. each element k represents a class inside the training set and it is computed by summing all the training points that belong to the class k. In line 5 the class hypervector $\mathcal{C}_H$ is normalized. In lines 6-8, an approximation of the cosine similarity (Eq.3) is then computed between each class hypervector C_{H_k} and the training hypervector H that is generated from the training datapoint.

$$cos(\mathcal{C}_k, \mathcal{H}) \propto \langle \mathcal{C}_k, \mathcal{H} \rangle \tag{3}$$

The equation 3 assumes that the class hypervector $\mathcal{C}_k$ is normalized. We removed the $\|\mathcal{H}\|$ since it is common to the cosine similarity between the training hypervector and all the class hypervector and hence not necessary for classification. This approximation makes the gradient calculation easier. A set of N_x probabilities with K dimensionality (K refers to the number of classes) $q_{i,k}$ is computed from this using the softmax function. A categorical cross-entropy loss $\mathcal{L}$ over the N_x training datapoints is computed in line 8 using the set of probabilities generated before and their corresponding true labels $y_{i,k}$ ($y_{i,k}$ stands for the i'th element of the training set and k is the k'th position of the one hot representation of y_i).

The gradient of $\mathcal{L}$ is then back-propagated through the $\mathcal{W}_{cnn}$ and $\mathcal{W}_d$ weights as described in lines 9-10.

Algorithm 1 DeepER-HD training System

1: $\eta = 10^{-3}$, $N_{epochs} = 300$, $\mathcal{W}_{cnn} = \mathcal{N}(0,1)$,$\mathcal{W}_d = \mathcal{N}(0,1)$
2: **for** 1 to N_{epochs} **do**
3: $\mathcal{H}_T = \mathcal{W}_d * \mathcal{M}_{cnn}(X, \mathcal{W}_{cnn})$
4: $\mathcal{C}_H = \mathcal{W}_d * \mathcal{M}_{cnn}(C, \mathcal{W}_{cnn})$
5: $\mathcal{C}_H = \frac{\mathcal{C}_H}{\|\mathcal{C}_H\|}$
6: **for** i from 1 to N_x **do**
7: **for** k from 1 to K **do**
8: $q_{i,k} = \frac{e^{\mathcal{H}_i \cdot \mathcal{C}_{H_k}^T}}{\sum_{j=0}^{K} e^{\mathcal{H}_i \cdot \mathcal{C}_{H_j}^T}}$
9: $\mathcal{L} = -\sum_{i=0}^{N_x}\sum_{k=0}^{K} y_{i,k} log(q_{i,k})$
10: $\mathcal{W}_{cnn} \leftarrow \mathcal{W}_{cnn} - \eta \frac{\partial \mathcal{L}}{\partial \mathcal{W}_{cnn}}$
11: $\mathcal{W}_d \leftarrow \mathcal{W}_d - \eta \frac{\partial \mathcal{L}}{\partial \mathcal{W}_d}$

B. Error Model

Modern systems suffer from errors in hardware function due to operation at low supply voltages and high clock speeds where hardware performance is stretched to its limits. Existing studies on DNN showed that the memory cells are more vulnerable to soft and timing error than the logic cells [20], [21]. Indeed, according to [20] SRAM are more exposed to large delay variation (timing error) since they use smaller devices than logic cells and there are vast number of cells in each chip making the probability of an outlier higher. In our study, we consider soft and timing errors in the SRAM of the associative memory when storing and retrieving the class hypervector and soft errors in the output of the CNN module of the DeepER-HD preprocessor. Such error in the associative memory are assumed to be caused by radiation, cosmic rays, and timing errors such as induced by voltage scaling, signal coupling or power and ground bounce [22]. Soft errors are modeled using transient bit flips as in [21]. Protecting the associative memory used to store class hypervectors against errors is thus crucial to trustworthy HDC operation [3], [15]. For evaluating the resilience of DeepER-HD, we used the Single-Bit-Flip (SBF) [20], [21] model applied to the numerical (binary) representations of the hypervector elements.

C. Error Resilience

Algorithm 2 describes the error-resilient HDC training approach. It relies on *restricting the ranges of hypervector elements* under soft errors through a *range restriction clipping* operation. It takes as input, bounds on the ranges of elements of the class hypervectors (i.e, **Min** and **Max**) derived from analysis of the hypervectors of the baseline HDC system (trained without the preprocessor of Fig. 1 and without range restriction). A single Min and Max bound is used for all the hypervectors in all iterations of Algorithm 2 as this was found to be the most effective for resilience. The system (preprocessor+HDC) is retrained with range restriction for N_{range} epochs and the initial maximum accuracy Max_{acc} is initialized to zero. Line 4 introduces the function **GWt** which takes as an input the training set X_T, its corresponding classes y_T, the pre-trained CNN feature extractor weights $\mathcal{W}_{cnn}^t$, dense layer weights $\mathcal{W}_d^t$ and the class hypervectors $\mathcal{C}_H^t$. **GWt** then generates updated preprocessor CNN module weights $\mathcal{W}_{cnn}^{t+1}$, preprocessor dense layer weights $\mathcal{W}_d^{t+1}$ and class hypervectors $\mathcal{C}_H^{t+1}$ iteratively as stated in Algorithm 1. In line 5, the class hypervectors $\mathcal{C}_H^{t+1}$ are clipped using the **Min** and **Max** bounds and the class hypervectors $\mathcal{C}_{H_{clip}}^{t+1}$ are generated. In line 6, the older values of the CNN feature extractor weights, the dense layer weights, and the class hypervector weights are updated. Then, in line 7, the function **Eval** is used with updated weights and the clipped class hypervectors on the validation set X_V and their corresponding labels y_V to calculate the validation accuracy Acc_{val}. Finally, the clipped class hypervectors, the CNN feature extractor weights, and the dense layer weights that correspond to the maximum validation accuracy Acc_{val} are stored. During Inference, in line-11 we look at all the class hypervector from 1 to K and clip them using the **Min** and **Max** bounds (line-12). Then, in line-13 we compute the cosine similarity between the class hypervector C_k and the query hypervector H_q. The class with the highest similarity is then the predicted class (line 14-16).

Algorithm 2 Range Restriction Filter HDC system

1: **procedure** RANGE RESTRICTION TRAINING(**Min**, **Max**)
2: $N_{range} = 100$, $Max_{acc} = 0$
3: **for** t from 1 to N_{range} **do**
4: $\mathcal{W}_{cnn}^{t+1}, \mathcal{W}_d^{t+1}, \mathcal{C}_H^{t+1} \leftarrow$ **GWt**$(X_T, \mathcal{C}_H^t, \mathcal{W}_{cnn}^t, \mathcal{W}_d^t)$
5: $\mathcal{C}_{H_{clip}}^{t+1} = Clip\left(\mathcal{C}_H^{t+1}, \textbf{Min}, \textbf{Max}\right)$
6: $\mathcal{W}_{cnn}^t, \mathcal{W}_d^t, \mathcal{C}_H^t \leftarrow \mathcal{W}_{cnn}^{t+1}, \mathcal{W}_d^{t+1}, \mathcal{C}_{H_{clip}}^{t+1}$
7: $Acc_{val} =$ **Eval**$(X_V, y_V, \mathcal{C}_H^t, \mathcal{W}_{cnn}^t, \mathcal{W}_d^t)$
8: **if** $Acc_{val} > Max_{acc}$ **then**
9: $\mathcal{W}_{cnn}^*, \mathcal{W}_d^*, \mathcal{C}_H^* = \mathcal{W}_{cnn}^t, \mathcal{W}_d^t, \mathcal{C}_H^t$
10: **procedure** RANGE RESTRICTION INFERENCE
11: **for** k from 1 to K **do**
12: $C_k = Clip(C_k, \textbf{Min}, \textbf{Max})$
13: $\delta_k = cos(C_k, H_q)$
14: **if** $\delta_k > \delta_{max}$ **then**
15: $\delta_{max} = \delta_k$
16: $\tilde{y} = k$

Choice of the range bounds (i.e, **Min** and **Max**) involves tradeoffs, since a narrow range results in loss of performance (i.e. low testing/inference accuracy) and wide range degrades system error resilience. Broadly, best performance was obtained by increasing the range bounds for lower accuracy HDC systems and vice-versa.

V. EVALUATION

A. Experimental setup

In this section we validate the use of trained convolutional feature extractors in DeepER-HD against regular HDC models [23], followed by comparisons of the error resilience of both DeepER-HD and regular HDC performance with and without

the range-filtering based resilience approach. The different dataset test cases and their characteristics for DeepER-HD and the baseline HDC system [2] used in our experiments are summarized in Table I.

After fine-tuning and freezing DeepER-HD model parameters, the range filter-based resilience framework is evaluated using soft-error injection experiments. The class hypervectors are encoded in 32-bits and errors are injected into the binary representation of each hypervector element with an error rate that varies in a logarithmic scale between 10^{-10} and 10^{-2}.

The framework of Section IV-C using a clipping filter was applied to the class hypervectors to enhance the error resilience of DeepER-HD (combination of Algorithms 1 and 2). The clipping filter was also applied to the baseline HDC model (not trained with range restriction) for comparison. The range value is determined empirically from the baseline system hypervectors as discussed earlier in Section IV-C. A range restriction filter is also applied to the output of the preprocessor system's feature extractor to enhance error resilience. DeepER-HD and this error resilience framework were implemented in hardware using Xilinx Vivado High-Level Synthesis on a Xilinx Zynq UltraScale+ MPSoC ZCU104 FPGA.

TABLE I: Datasets Description

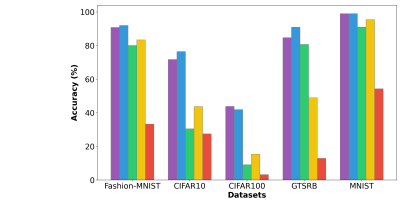
Application	Name	Train Size	Test Size	Features	# classes
Voice Recognition	ISOLET	6238	1599	617	26
Human Activity Recognition	UCIHAR	7767	3162	561	6
Medical Diagnosis	CARDIO	1424	702	23	3
Face Recognition	FACE	1408	694	2352	2
Image Classification	MNIST	60k	10k	28x28x1	10
	Fashion MNIST	60k	10k	28x28x1	10
	CIFAR10	50k	10k	32x32x3	10
	CIFAR100	50k	10k	32x32x3	100
	GTSRB	39k	12.6k	32x32x3	43

B. Performance of DeepER-HD

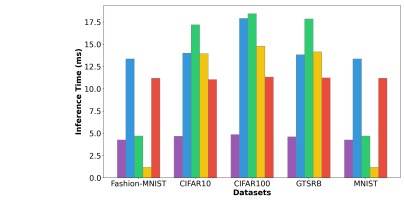
The figure 2(a) and 2(b) shows a comparison between DeepER-HD with 2 convolutional layers and an HD encoder (D=1k), a regular CNN with multiple convolutional and max-pooling layers , the Baseline HD [23] with a dimensionality (D = 10k) , a linear support vector machine (SVM) and 100 estimators Adaboost algorithm in term of accuracy and inference latency. DeepER-HD has a comparable accuracy to a tiny CNN for classifying image dataset (e.g, Fashion-MNIST, CIFAR10, GTSRB) and even better accuracy than a CNN (CIFAR100). It outperforms all the other machine learning technique in terms of accuracy (i.e, SVM and AdaBoost). Due to the use of a feature extractor, the DeepER-HD has better accuracy than the Baseline-HD. In term of latency, DeepER-HD has lower overhead than the CNN and other machine learning techniques (i.e, SVM and Adaboost) and is also faster than the Baseline-HD due to its lower dimensionality. As a conclusion, DeepER-HD has as the best Accuracy-latency balance compared to standard machine learning techniques.

C. Error resilience

Model resilience to soft errors in associative memory was assessed on the baseline and DeepER-HD. The use of the range filter method of Section IV-C was compared against the base case of no error resilience method applied for



(a) Accuracy comparison

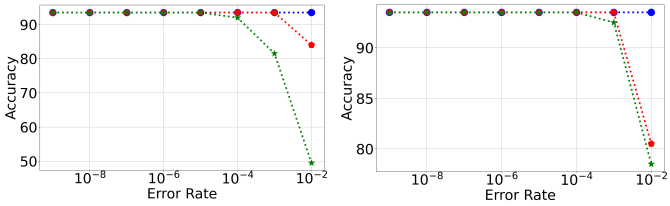


(b) Inference latency comparison

(c)

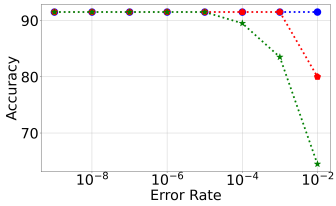
Fig. 2: DeepER-HD Inference latency and Accuracy comparison with State-Of-The-Art

both the baseline and DeepER-HD. In these experiments,the failure error rate is defined as the error rate that causes a 5% accuracy drop from the nominal accuracy. The relative difference between the failure error rate of two technique is defined as the error resilience of one method compared to the other. Figure 3 shows, on a semi-logarithmic scale of error rate, the accuracy of the HDC baseline with our resilience method and the baseline technique [15] on different datasets (i.e, UCIHAR, ISOLET, and CARDIO).



(a) ISOLET

(b) UCIHAR



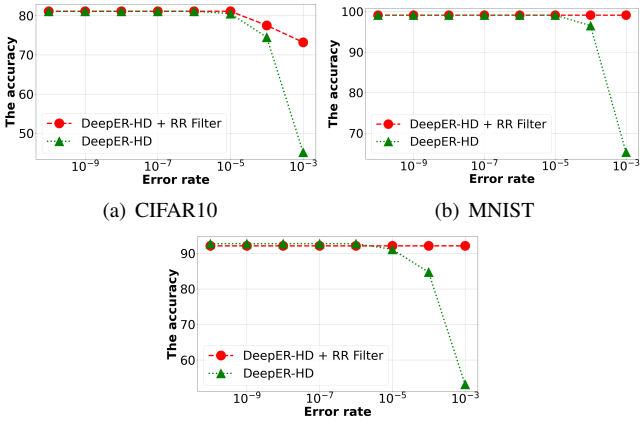
(c) CARDIO

(d)

Fig. 3: Error resilience assessment for HDC baseline

For example in figure 3(a) our technique is one order of magnitude more error resilient than the baseline [15] and three order of magnitude more error resilient than the faulty model. In figure 3(b), our method is one order of magnitude more error resilient than the baseline [15] and two order of

magnitude more robust to error than the faulty model. Finally, in figure 3(c), our technique is respectively one and two order of magnitude more error resilient than the baseline [15] and the faulty model.



(a) CIFAR10 (b) MNIST

(c) FASHION MNIST

Fig. 4: Error resilience assessment for DeepER-HD (errors in the class hypervector)

The range restriction filter (see algorithm 2) was also applied to DeepER-HD with the CNN encoder. Figures 4 show the performance of our range restriction filter used in DeepER-HD for soft errors in the class hypervector (in associative memory) on the MNIST, FashionMNIST, CIFAR10, and GTSRB datasets compared to the unprotected DeepER-HD model. The range restriction filter showed an error resilience of two order of magnitude compared to the faulty model when applied on MNIST and Fashion-MNIST datasets. The error resilience drops to one order of magnitude for more complex dataset like CIFAR10. The level of resilience drops when dealing with more complex classification tasks (i.e, CIFAR10 or GTSRB) due to the tradeoff between tight range bounds (for resilience) and accuracy (requiring looser bounds).

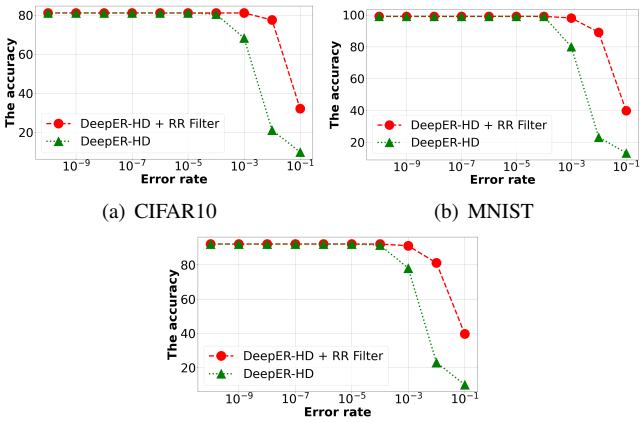
The resilience of DeepER-HD was also assessed against soft errors injected into the CNN feature extractor.

A range restriction filter is placed after the feature extractor module to prevent accuracy degradation due to soft errors inside the feature extractor. Figure 6 shows the accuracy of DeepER-HD with and without the range restriction filter after injecting soft errors over a range of error rates. Our technique is two order of magnitude more robust to error than the faulty model.

D. Hardware Validation

Here we assess the performance in terms of accuracy and the overhead (power consumption, inference time, resource utilization) of DeepER-HD on the Zynq UltraScale+ MPSoC ZCU104 FPGA against the baseline HDC system of [2]. Table II summarizes the overhead and the accuracy of DeepER-HD compared to the baseline in hardware validation on FPGA.

Table II shows that the DeepER-HD achieves better accuracy for the CIFAR10 (+135%), MNIST (+7%), and Fashion-MNIST (+11.6%) classification tasks. However, the DeepER-HD model incurs higher overhead compared to the due to CNN



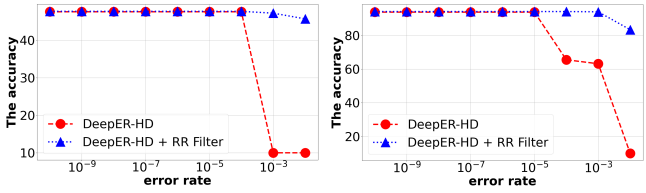
(a) CIFAR10 (b) MNIST

(c) FASHION MNIST

Fig. 5: Error resilience assessment for DeepER-HD (errors in the feature extractor)

feature extractor usage. In all cases except the CIFAR10 classification task, the baseline consumed less memory (BRAMs -12.87%) and had a slightly lower inference time (-0.75%). The high dimensionality of the CIFAR10 dataset (32x32x3) leads to high matrix multiplication dimensionality (i.e the encoder of the HDC baseline) compared to DeepER-HD, implying that as dataset dimensionality rises in image classification, DeepER-HD is more cost-effective.

The error resilience of DeepER-HD has also been assessed on the Zynq UltraScale+ MPSoC ZCU104 Board. Single-bit flips were injected inside the class hypervector. Table III shows the overhead of the regular DeepER-HD model compared to its error-resilient version. As shown in Table III, the range



(a) CIFAR10 (b) MNIST

Fig. 6: Hardware assessment Error resilience for DeepER-HD

restriction filter adds almost no energy overhead on MNIST data and adds 8% energy overhead for CIFAR10 dataset. It turns out that the encoding system is the most time-intensive part of the DeepER-HD model with (97 % and 87 %), while similarity checking represents (2.9 % and 13.25 %) of the inference time over CIFAR10 and MNIST respectively. The range restriction filter adds negligible time overhead for both cases. Figure 6 shows the accuracy of regular DeepER-HD and error-resilient DeepER-HD after injecting randomized single-bit flips inside the class hypervectors. The range restriction filter provides two order of magnitude of error resilience compared to the faulty model.

VI. Conclusion

Hyperdimensional Computing is an emerging paradigm in machine learning with low computational overhead and inference time. We proposed DeepER-HD, an deep learning

TABLE II: DeepER-HD performance hardware assessment

Dataset	Model	Resource utilization				Inference time (us)	Power (W)	Energy (mJ)	Accuracy (%)
		BRAM	FF	LUT	DSP				
MNIST	DeepER-HD	178	49252	36432	331	55.5 (-12%)	3.732	0.207	97.7 (+7%)
	Baseline	83.5	14072	12893	11	49.5	2.868	0.141 (-31%)	91
Fashion MNIST	DeepER-HD	178	49252	36432	331	55.5 (-12%)	3.732	0.207	89.38 (+11.6%)
	Baseline	83.5	14072	12893	11	49.5	2.868	0.141 (-31%)	80.03
CIFAR10	DeepER-HD	182.5	46329	34824	311	54.3 (+0.75%)	3.698	0.2	71.7 (+135%)
	Baseline	206.5	14098	13055	11	71.72	2.89	0.152 (-24%)	30.5

TABLE III: DeepER-HD error resilience hardware assessment

Dataset	model	Resource utilization				Power (W)	Inference time (ms)	Energy
		BRAM	FF	DSP	LUT			
MNIST	DeepER-HD	130.5	36580	336	29996	1.004	1.539	1.545
	Error Resilient DeepER-HD	130.5	37127	336	30323	1.011	1.539	1.555
	Difference	**+0.0%**	**+1.5%**	**+0.0%**	**+1.1%**	**+0.69%**	**+0.0%**	**+0.69%**
CIFAR10	DeepER-HD	210.5	36299	322	30029	0.937	7.007	6.545
	Error Resilient DeepER-HD	210.5	35417	322	30041	1.014	7.007	7.105
	Difference	**+0.0%**	**+0.32%**	**+0.0%**	**+0.04%**	**+8.21%**	**+0.0%**	**+8.21%**

inspired HDC model that outperforms state-of-the-art HDC systems with low additional overhead. An error resilience framework for DeepER-HD and traditional HDC was also presented. Future work aims to make DeepER-HD more cost-effective and explore further error resilience methodologies.

ACKNOWLEDGMENT

This research was supported by NSF Grant No. 2128149 and by the School of ECE at Georgia Institute of Technology.

REFERENCES

[1] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1(2):139–159, 2009.

[2] Lulu Ge and Keshab K Parhi. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine*, 20(2):30–47, 2020.

[3] Sizhe Zhang, Ruixuan Wang, Jeff Jun Zhang, Abbas Rahimi, and Xun Jiao. Assessing robustness of hyperdimensional computing against errors in associative memory : (invited paper). In *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 211–217, 2021.

[4] Mohsen Imani, Justin Morris, John Messerly, Helen Shu, Yaobang Deng, and Tajana Rosing. Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[5] Tony F. Wu, Haitong Li, Ping-Chen Huang, Abbas Rahimi, Jan M. Rabaey, H.-S. Philip Wong, Max M. Shulaker, and Subhasish Mitra. Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 492–494, 2018.

[6] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, 2017.

[7] Mohammad Yasser, Khaled F. Hussain, and Samia Abd El-Fattah Ali. An efficient hyperdimensional computing paradigm for face recognition. *IEEE Access*, 10:85170–85179, 2022.

[8] Peer Neubert, Stefan Schubert, and Peter Protzel. An introduction to hyperdimensional computing for robotics. *KI - Künstliche Intelligenz*, 33, 09 2019.

[9] Mohsen Imani, Samuel Bosch, Sohum Datta, Sharadhi Ramakrishna, Sahand Salamat, Jan M. Rabaey, and Tajana Rosing. Quanthd: A quantization framework for hyperdimensional computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10), 2020.

[10] Mohsen Imani, John Messerly, Fan Wu, Wang Pi, and Tajana Rosing. A binary learning framework for hyperdimensional computing. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 126–131, 2019.

[11] Shijin Duan, Yejia Liu, Shaolei Ren, and Xiaolin Xu. Lehdc: Learning-based hyperdimensional computing classifier. *arXiv preprint arXiv:2203.09680*, 2022.

[12] Peter Sutor, Dehao Yuan, Douglas Summers-Stay, Cornelia Fermuller, and Yiannis Aloimonos. Gluing neural networks symbolically through hyperdimensional computing. *arXiv preprint arXiv:2205.15534*, 2022.

[13] Arpan Dutta, Saransh Gupta, Behnam Khaleghi, Rishikanth Chandrasekaran, Weihong Xu, and Tajana Rosing. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, pages 281–286, 2022.

[14] Tao Yu, Yichi Zhang, Zhiru Zhang, and Christopher De Sa. Understanding hyperdimensional computing for parallel single-pass learning. *arXiv preprint arXiv:2202.04805*, 2022.

[15] Sizhe Zhang, Mohsen Imani, and Xun Jiao. Scalehd: robust brain-inspired hyperdimensional computing via adapative scaling. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[16] Chandramouli Amarnath, Mohamed Mejri, Kwondo Ma, and Abhijit Chatterjee. Soft error resilient deep learning systems using neuron gradient statistics. In *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2022.

[17] Elbruz Ozen and Alex Orailoglu. Boosting bit-error resilience of DNN accelerators through median feature selection. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(11):3250–3262, 2020.

[18] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–13. IEEE, 2021.

[19] Abbas Rahimi, Pentti Kanerva, and Jan M. Rabaey. A robust and energy-efficient classifier using brain-inspired hyperdimensional computing. ISLPED '16, page 64–69, New York, NY, USA, 2016. Association for Computing Machinery.

[20] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. Ares: A framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[21] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[22] Weiwei Shan, Longxing Shi, and Jun Yang. In-situ timing monitor-based adaptive voltage scaling system for wide-voltage-range applications. *IEEE Access*, 5:15831–15838, 2017.

[23] Alejandro Hernández-Cano, Namiko Matsumoto, Eric Ping, and Mohsen Imani. Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 56–61. IEEE, 2021.