2023 IEEE 41st International Conference on Computer Design (ICCD) | 979-8-3503-4291-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCD58817.2023.00045

HyperMetric: Robust Hyperdimensional Computing on Error-prone Memories using Metric Learning

Weihong Xu, Viji Swaminathan, Sumukh Pinge, Sean Fuhrman, and Tajana Rosing
University of California San Diego, La Jolla, CA 92093, USA
Email: {wexu, tajana}@ucsd.edu

***Abstract*—Hyperdimensional computing (HDC) is emerging as an efficient and robust computing paradigm that has strong resilience to various types of errors. The robustness of HDC makes it a good match for error-prone memory systems. In this work, we propose HyperMetric, a framework to develop highly robust and hardware-friendly HDC models. First, we propose HyperMetric training which is based on metric learning to optimize for high robustness. The experiments show that HyperMetric-trained HDC models deliver up to $17\times$ larger distance margin and 14.3% accuracy gain. Compared to state-of-the-art HDC algorithms OnlineHD [1] and HyDREA [2], HyperMetric ReRAM accelerator is $> 20\%$ more accurate for computing-in-memory (CIM) errors and $> 10\%$ more accurate for bit errors even in the face of variations. Furthermore, HyperMetric hardware is 35% more accurate in comparison with state of the art tinyHD [3] and GENERIC [4] accelerators in the face of $3\times$ ReRAM resistance variance, and 20% more accurate with BER of up to 20% due to voltage scaling while keeping a good balance between area, power, and processing latency.**

***Index Terms*—Hyperdimensional computing, metric learning, memory errors, hardware robustness.**

I. Introduction

Deep learning (DL) has shown excellent accuracy on various classification tasks. However, the high accuracy comes at the cost of expensive computation complexity and data movement, hindering DL's deployment on low-power embedded devices with constrained resources. ReRAM-based computing-in-memory (CIM) [5, 6] and voltage scaling techniques [7, 8] are the two effective solutions to obtain high energy efficiency for data-intensive DL algorithms. However, increasing energy efficiency while maintaining high inference accuracy for DNNs is challenging because both ReRAM-based CIM and voltage scaling schemes are inevitably error-prone. ReRAM has proven unreliable and incurs accuracy degradation when running DNNs [6, 9] due to imperfect memory devices and sensing circuits [9]. Meanwhile, aggressive voltage scaling helps reduce power consumption, but the number of bit errors increases while the DNN's inference accuracy drops exponentially [7, 8].

There are two popular approaches to address memory errors: **1. adding error correction (ECC) hardware** [5, 10], **2. enhancing the algorithm's robustness** [2, 6]. While adding ECC significantly improves the memory's reliability, it is not cost-effective due to the large implementation overhead. For example, CIM-SECDED [5] sacrifices 20% memory cells and requires additional 15% energy consumption for ECC, degrading both energy and hardware efficiency. Enhancing the algorithm's robustness is hardware-friendly without costly circuit modifications. However, the DNN models do not have good error resilience as $> 1\%$ error rate may bring catastrophic accuracy loss [7, 8].

In this work, we propose the HyperMetric framework to develop the error-robust and low-complexity classification algorithm as well as hardware implementation for error-prone memories based on the brain-inspired hyperdimensional computing (HDC) [11, 12]. HDC processes cognitive tasks in a lightweight manner [3, 13]: the inference can be performed via Hamming distance-based associative memory search, only requiring binary XOR and addition operations. HDC's unique computing pattern is efficient for both CIM and low-power applications [1, 13–15]. Meanwhile, HDC is error-resilience and can be easily integrated into existing memory architecture without additional modifications. The Hamming distance search shows great error robustness due to HDC's distributed nature of feature storage [11]. Our contributions include:

- We propose HyperMetric, a framework to derive robust and efficient HDC designs. HyperMetric uses metric learning [16] to optimize the Hamming distance margin, which is improved by 2-17$\times$ compared to state-of-the-art OnlineHD [1] and HyDREA [2]. The large Hamming distance margin significantly improves model error resilience (10-35$\%$ less accuracy loss) and classification accuracy (up to 14.3$\%$ gain).
- We present the efficient ASIC design for HyperMetric, achieving excellent robustness and balance between latency, power, and overhead compared to existing HDC designs (tinyHD [3] and GENERIC [4]).

II. Preliminary

A. Error-prone Memories

We consider two types of typical memory errors in this work: **Type 1 Readout Bit Errors** are mainly resulted from scaling down the voltage of memories [7, 8]. Though aggressive reduction for on-chip memory's voltage is beneficial for achieving higher energy efficiency for data-intensive DNN inference tasks, the lower readout voltage inevitably injects errors into the readout data from voltage-scaled memories. As a result, the errors create significant accuracy degradation. We emulate the readout error patterns using the random bit flipping model with a bit error rate (BER) given in [7]. **Type 2 CIM Errors** exist in the silicon-proven 1T1R ReRAM architecture [5]. ReRAM suffers from various types of hardware imperfections, including limited resistance ratios, variable resistance distributions, imperfect sense amplifiers, and leakage current [9]. These imperfections cause CIM errors and result in severe accuracy loss for DNN inference on ReRAM.

2576-6996/23/$31.00 ©2023 IEEE
DOI 10.1109/ICCD58817.2023.00045

B. Hyperdimensional Computing (HDC)

The HDC algorithm consists of the encoding, training, and inference steps as follows:

1) Encoding: Consider a feature vector $\mathbf{x} \in \mathbb{R}^F$. The goal of HDC encoding is to convert $\mathbf{x}$ into a D-dimensional binary hypervector (HV) $\mathbf{H} \in \{+1, -1\}^D$ using the signed random projection (RP) with a matrix $\mathbf{R} \in \mathbb{R}^{D \times F}$:

$$\mathbf{H} = \text{sign}(\mathbf{R} \cdot \mathbf{x}). \tag{1}$$

2) Training and Retraining: For each class j, we denote $\mathbf{H}_i^j$ as the encoded HV for input feature $\mathbf{x}_i$, belonging to j-th class. Assuming there are k encoded HVs for for each class j, the associated class HV $\mathbf{C}_j \in \{+1, -1\}^D$ is computed as:

$$\mathbf{C}_j = \text{sign}(\sum_{n=1}^{k} \mathbf{H}_n^j), \tag{2}$$

which means that we are simply adding together all encoded HV for each class. The HDC model is then built using a collection of all class HVs $\mathbf{C}_j$. A major advantage of HD is the simplicity of training, as it can reach high accuracy with a single pass through the training data.

The accuracy of the HDC model can then be improved through the retraining process, where each feature vector is encoded again and compared to each class HV. The prediction is made from the most similar class HV. Then for each incorrect prediction, we update the model as follows:

$$\begin{cases} \mathbf{C}_j &= \mathbf{C}_j + \alpha \cdot \mathbf{H}^j, \\ \mathbf{C}_k &= \mathbf{C}_k - \alpha \cdot \mathbf{H}^j, \end{cases} \tag{3}$$

where α denotes the learning rate applied to update class HVs. The intermediate data of updated class HVs are non-binary. For the binary HDC model, the class HVs are finally binarized using the sign function after the retraining process.

3) Inference: The inference step finds the most similar class HV $\mathbf{C}_j$ to the encoded HV $\mathbf{H}$ as: $\arg\min_j \text{Hamming}(\mathbf{H}, \mathbf{C}_j)$, where the HDC prediction is made based on the class that generates the smallest Hamming distance.

III. HyperMetric Algorithm and Hardware Design

A. HyperMetric Training

The HDC noise robustness theory (Theorem 10) in [12] demonstrates that increasing the Hamming distance margin (distance difference between the correct class and other classes) can improve the HDC error resilience. In Fig. 1, we propose HyperMetric training to improve HDC's model robustness based on deep metric learning [16]. During the retraining phase, existing HDC models first encode the query and then the class HVs are updated based on the correctness of prediction (Eq. (3)). Instead, HyperMetric optimizes both class HVs and HDC encoder to maximize the Hamming distance margin. HyperMetric uses the following CosFace loss [16] modified based on Softmax to perform the stochastic gradient descent (SGD)-based training. For each batch of N training samples, the CosFace loss is:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{e^{s \cdot \left(\cos(\theta_{y_i,i}) - m\right)}}{e^{s \cdot \left(\cos(\theta_{y_i,i}) - m\right)} + \sum_{j \neq y_i} e^{s \cdot \cos(\theta_{j,i})}}, \tag{4}$$

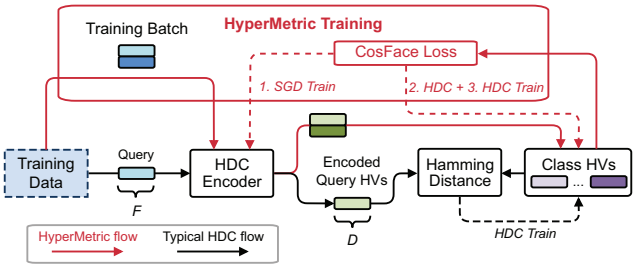

Fig. 1: Comparison for HyperMetric and typical HDC.

where $\cos(\theta_{j,i}) = \mathbf{C}_j^* \cdot \mathbf{x}_i^*$ is the cosine angle between the normalized j-th class $\mathbf{C}_j^*$ and the normalized i-th feature vector $\mathbf{x}_i^*$. m controls the penalty threshold of cosine margin while s is the scaling parameter that impacts the classification performance. We replace the original linear layer with normalized HDC class HVs $\mathbf{C}^*$. This allows us to use SGD to optimize both HDC encoder weight $\mathbf{R}$ and class HVs $\mathbf{C}$.

HyperMetric training mainly consists of three steps: 1) Use the modified CosFace loss in Eq. (4) with SGD to optimize the HDC encoder weights $\mathbf{R}$ while freezing the class HVs $\mathbf{C}$. 2) Freeze the encoder weights $\mathbf{R}$ and use the HDC retraining in Eq. (3) to update only the class HVs $\mathbf{C}$. 3) Update both encoder weights $\mathbf{R}$ and class HVs $\mathbf{C}$. The other modification is that HyperMetric's training needs the soft gradient information. Thus, the signed random projection in Eq. (1) is revised to a differentiable encoding function: $\mathbf{H} = \tanh(\mathbf{R} \cdot \mathbf{x})$.

B. Hardware Architecture and Implementation

1) Approximate Encoding: The HDC encoding in Eq. (1) requires $\mathcal{O}(D \cdot F)$ complexity. Previous works [13] show it is critical to reducing the encoding overhead for better efficiency. We propose the approximate encoding to reduce the encoding overhead. The Kronecker product (KP) [17] is used to decompose the encoding weight matrix $\mathbf{R}$ into the summation of r KPs of sub-matrices $\mathbf{A}_i$ and $\mathbf{B}_i$ as:

$$\mathbf{R} \approx \left(\sum_{i=1}^{r} \mathbf{A}_i \otimes \mathbf{B}_i \right) \cdot \mathbf{x} \approx \sum_{i=1}^{r} \mathbf{B}_i \cdot \text{vec}(\mathbf{x}) \cdot \mathbf{A}_i^T, \tag{5}$$

where $\otimes$ denotes the KP operation. $\mathbf{A}_i \in \mathbb{R}^{d_{1,i} \times f_{1,i}}$ and $\mathbf{B}_i \in \mathbb{R}^{d_{2,i} \times f_{2,i}}$, where the submatrix dimension satisfies: $\prod_i d_{1,i} \cdot d_{2,i} = D$ and $\prod_i f_{1,i} \cdot f_{2,i} = F$. $\text{vec}(\cdot)$ reshape the input vector to a new matrix in the column-major order.

The KP-based approximate encoding effectively reduces the encoding overhead in terms of computational complexity and the memory space. Our evaluation shows 13-20$\times$ computation complexity reduction and 180$\times$ compression ratio. The total number of summations r (called rank) determines the reconstruction quality. A higher rank leads to less reconstruction error as more free parameters can be used to reconstruct $\mathbf{R}$. But it also reduces the compression effect. We discuss the optimal rank value to balance quality and efficiency in Section IV.

2) Retraining: To obtain better performance and faster training convergence, the sub-matrices $\{\mathbf{A}_i, \mathbf{B}_i\}$ are initialized to minimize the reconstruction error given a rank value r. This is achieved by solving the following equation that generates the minimum L_2 distance to the full-precision encoding weight

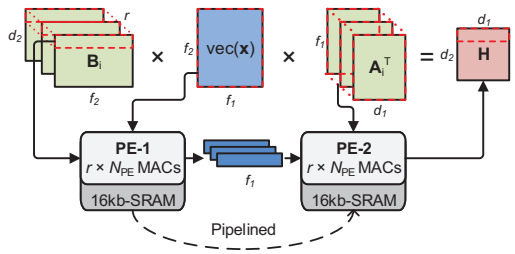

Fig. 2: Pipelined hardware architecture for the KP-based approximate encoder in HyperMetric.

matrix $\mathbf{R}$: $\arg\min_{\mathbf{A}_i,\mathbf{B}_i} \|\mathbf{R} - \sum_{i=1}^{r} \mathbf{A}_i \otimes \mathbf{B}_i\|^2$, which can be solved effectively based on the singular value decomposition (SVD) of $\mathbf{R}$ according to [17]. The KP-based approximate encoding introduces performance degradation to the full-precision model. Thus, an additional retraining phase is needed to restore the accuracy and Hamming distance margin. The retraining follows the identical process as HyperMetric training.

3) Hardware Design: We propose a pipelined encoder architecture in Fig. 2 to efficiently realize the approximate encoding in Eq. (5). The encoder architecture consists of 2 processing element (PE) arrays, each with $r \times N_{\text{PE}}$ MACs, r N_{PE}-input adder trees, and 16kb SRAM buffer, respectively. The SRAM in each PE stores the KP sub-matrices with 4-b quantization. The KP-based approximate encoding is processed in a two-stage pipelined manner to maximize the hardware utilization: the first array `PE-1` computes each row of $\mathbf{B}_i \cdot \text{vec}(\mathbf{x})$ while the second array `PE-2` computes each row of encoded output $\mathbf{H}$ based on the results from `PE-1`.

`PE-1` parallelizes the computation along the rank r dimension in its $r \times N_{\text{PE}}$ MACs. Each cycle `PE-1` computes the partial sum for N_{PE} elements at the same row of matrix $\mathbf{B}$ and the corresponding N_{PE} elements from the reshaped input feature vec$(\mathbf{x})$. `PE-1` consumes $f_1 \cdot \lceil \frac{f_2}{N_{\text{PE}}} \rceil$ cycles to generate one f_1-dimensional row of $\mathbf{B}_i \cdot \text{vec}(\mathbf{x})$ with r ranks. Then `PE-2` receives the row data from `PE-1` and reuses them to multiply with each $\mathbf{A}_i^T$, respectively. The output from `PE-2` is r-rank $\mathbf{B}_i \cdot \text{vec}(\mathbf{x}) \cdot \mathbf{A}_i$ data and needs to be accumulated to generate the partial encoded HV $\mathbf{H}$. The encoded HV is sequentially generated row by row. `PE-2` requires $d_1 \cdot \lceil \frac{f_1}{N_{\text{PE}}} \rceil$ cycles to generate each row for $\mathbf{H}$.

The computations of `PE-1` and `PE-2` are overlapped. To balance latency and hardware complexity, we set $N_{\text{PE}} = 16$ and $r = 4$, meaning that the proposed encoder design supports HDC dimension $D = 1024$ with input feature dimension $F = 1024$. HyperMetric hardware supports the KP rank value $r = 4$ because the rank $r = 4$ achieves a good trade-off between algorithm performance and hardware overhead.

IV. Evaluation

A. Methodology

HyperMetric Algorithm. The HyperMetric training algorithm is implemented using PyTorch. The training uses a batch size of 256 and the SGD optimizer with a learning rate $lr = 1e^{-3}$.

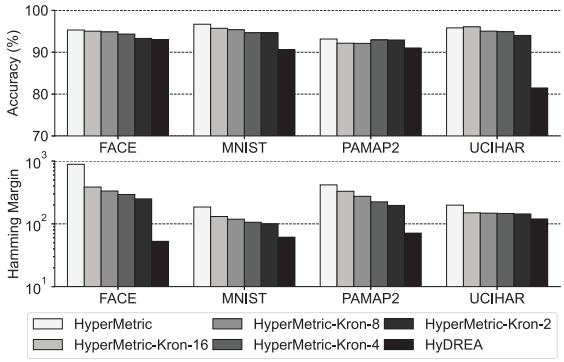

Fig. 3: Accuracy and Hamming distance margin (log scale) for Kronecker product encoding with rank values from 2 to 16 and HDC dimension $D = 1024$.

Each HDC is trained for 35 epochs using either metric learning or HDC retraining. The default HDC dimension D is 1024.

Hardware Modeling. The encoder hardware of HyperMetric is implemented using Verilog HDL and synthesized by Synopsys Design Compiler using TSMC 40nm technology node. The clock frequency is 500MHz. The ReRAM parameters are extracted from foundry's ReRAM [5]. We use the log-norm distribution [9] to fit the resistance distribution. We do not assume any write verification schemes that reduce the resistance variance. We build an in-house simulator based on DL-RSIM [9] to emulate the CIM behavior of ReRAM. The reference current of sense amplifier is the midpoint of two neighbors. We assume that the number of activated WLs is $N_{\text{WL}} = 16$. For SRAM voltage scaling, we inject random bit flipping errors into the readout data based on the BERs [7, 8].

TABLE I: Specifications of evaluated datasets.

Dataset	Application	Class C	Feature F	Train Size	Test Size
MNIST [18]	Image classification	10	784	60,000	10,000
UCIHAR [19]	Activity recognition	12	561	6,213	1,554
FACE [20]	Face detection	2	608	1,913	213
PAMAP2 [21]	Activity recognition	5	27	16,384	16,384

Baselines and Datasets. We compare HyperMetric to four state-of-the-art HDC algorithms (HyDREA [2] and OnlineHD [1]) and hardware designs (tinyHD [3] and GENERIC [4]). We use four real-world datasets for various applications as in Table I.

B. HyperMetric Algorithm and Hardware Evaluation

Approximate Encoding. Fig. 3 shows the impact of KP-based approximate encoding on the inference accuracy and Hamming distance margin. HyperMetric-Kron-r represents the KP-based encoding with rank r and the weights quantized to 4-b. Full-precision HyperMetric and HyDREA are the baselines for comparison. The approximation of KP and quantization introduce degradation to both accuracy and Hamming distance margin. Up to 2% accuracy loss and 72% Hamming margin decrease are observed using HyperMetric-Kron-2. The degradation is less significant for higher rank r because more free parameters can be tuned to restore the full-precision performance. However, HyperMetric with approximate encoding still outperforms HyDREA [2] in terms of accuracy and Hamming margin.

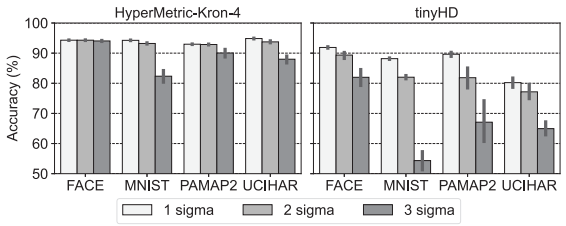

Fig. 4: HyperMetric and tinyHD [3] accuracy on ReRAM.

TABLE II: HyperMetric vs. HDC ASICs [3, 4] in 40nm node.

Design	tinyHD [3]	GENERIC [4]	HyperMetric
Enc. Algorithm	Cyclic RP	Optimized level-ID	Metric learning + RP
HD dim. D	4000	4000	1024
Area (mm^2)	0.025	0.297	0.099
Power (mW)	15.351	0.735	5.774
Latency (cycles)	12250	256	1568
Accuracy	93.9%	94.0%	96.1%
Error resilience	Low	Low	High

ReRAM-based CIM Evaluation. Fig. 4 shows the impact of ReRAM's resistance variance (1-3× sigma) on inference accuracy. We use the HyperMetric-Kron-4 model and adopt tinyHD [3] as the baseline. HyperMetric achieves much better error resilience compared to tinyHD: the inference accuracy on FACE and PAMAP2 datasets receives the least influence because the Hamming distance margins on these datasets are highest. In the worst case 3× variance, HyperMetric's 13% accuracy loss is much lower than tinyHD's 35% on MNIST.

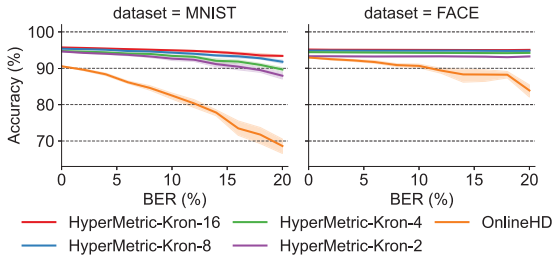

Fig. 5: Accuracy vs. BERs for HyperMetric vs. OnlineHD [1].

BER Evaluation for Voltage Scaling. Fig. 5 shows the impact of bit errors on the accuracy when scaling SRAM voltages [7, 8]. OnlineHD [1] is considered as the baseline. OnlineHD incurs over 20% accuracy loss on MNIST when BER increases to 20%. HyperMetric with approximate encoding shows $< 10\%$ loss in the worst case. For FACE dataset, negligible accuracy loss is observed due to the much larger Hamming margin compared to MNIST.

Hardware comparison with other HDC designs. For HyperMetric's hardware implementation, we choose HyperMetric-Kron-4 with HDC dimension $D = 1024$ since it provides good accuracy loss as well as Hamming distance margin while not requiring costly overhead. The key hardware parameters of HyperMetric are listed in Table II. Two state-of-the-art HDC encoder designs on ASIC, tinyHD [3] and GENERIC [4], are used for comparison. We only include the hardware components related to HDC encoding for fair comparison. The encoding latency and accuracy are calculated on dataset MNIST [18]. While HyperMetric is 4× larger, it achieves 2.7× power saving and 8× encoding latency reduction as compared to tinyHD [3]. GENERIC [4] achieves better power efficiency and shorter encoding latency at the cost larger area. HyperMetric achieves a good balance between area, power and latency. Notably, HyperMetric delivers the best accuracy and error resilience among the three designs.

V. Conclusion

In this work, we present HyperMetric's metric learning-based training for obtaining more accurate HDC models with enhanced error resilience. Then we propose the efficient ASIC hardware to implement HyperMetric. The experiments show that the HDC model and hardware optimized by HyperMetric have significantly strengthened error resilience and hardware efficiency compared to state-of-the-art HDC baselines [1–3].

Acknowledgements

This work is partially supported by the SRC JUMP 2.0 PRISM Center, and National Science Foundation (NSF) grants #1826967, #1911095, #2052809, #2112665, #2112167, and #2100237.

References

[1] A. Hernandez-Cane *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *DATE*, 2021, pp. 56–61.
[2] J. Morris *et al.*, "Hydrea: Towards more robust and efficient machine learning systems with hyperdimensional computing," in *DATE*, 2021, pp. 723–728.
[3] B. Khaleghi *et al.*, "tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications," in *DATE*. IEEE, 2021, pp. 408–413.
[4] ——, "Generic: highly efficient learning engine on edge using hyperdimensional computing," in *DAC*, 2022, pp. 1117–1122.
[5] B. Crafton *et al.*, "Cim-secded: A 40nm 64kb compute in-memory rram macro with ecc enabling reliable operation," in *A-SSCC*, 2021, pp. 1–3.
[6] J. Lin *et al.*, "Rescuing rram-based computing from static and dynamic faults," *IEEE TCAD*, vol. 40, no. 10, pp. 2049–2062, 2020.
[7] A. Di Mauro *et al.*, "Always-on 674μ w@ 4gop/s error resilient binary neural networks with aggressive sram voltage scaling on a 22-nm iot end-node," *IEEE TCAS-I*, vol. 67, no. 11, pp. 3905–3918, 2020.
[8] L. Yang and B. Murmann, "Sram voltage scaling for energy-efficient convolutional neural networks," in *ISQED*, 2017, pp. 7–12.
[9] W.-T. Lin *et al.*, "Dl-rsim: A reliability and deployment strategy simulation framework for reram-based cnn accelerators," *ACM TECS*, vol. 21, no. 3, pp. 1–29, 2022.
[10] B. Crafton *et al.*, "Statistical optimization of compute in-memory performance under device variation," in *ISLPED*, 2021, pp. 1–6.
[11] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, pp. 139–159, 2009.
[12] A. Thomas *et al.*, "A theoretical perspective on hyperdimensional computing," *Journal of Artificial Intelligence Research*, vol. 72, pp. 215–249, 2021.
[13] W. Xu *et al.*, "Fsl-hd: Accelerating few-shot learning on reram using hyperdimensional computing," in *DATE*, 2023.
[14] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, 2016, pp. 64–69.
[15] S. Zhang *et al.*, "Scalehd: Robust brain-inspired hyperdimensional computing via adapative scaling," in *ICCAD*, 2022, pp. 1–9.
[16] H. Wang *et al.*, "Cosface: Large margin cosine loss for deep face recognition," in *CVPR*, 2018, pp. 5265–5274.
[17] M. G. A. Hameed *et al.*, "Convolutional neural network compression through generalized kronecker product decomposition," in *AAAI*, vol. 36, no. 1, 2022, pp. 771–779.
[18] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[19] D. Anguita *et al.*, "A public domain dataset for human activity recognition using smartphones," in *International European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2013, pp. 437–442.
[20] A. Angelova *et al.*, "Pruning training sets for learning of object categories," in *CVPR*, vol. 1, 2005, pp. 494–501.
[21] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *International Symposium on Wearable Computers*, 2012, pp. 108–109.