Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)
Sister Conferences Best Papers Track

Algorithm-Hardware Co-Design for Efficient Brain-Inspired Hyperdimensional Learning on Edge (Extended Abstract)*

Yang Ni[1], **Yeseong Kim**[2], **Tajana Rosing**[3] and **Mohsen Imani**[1]
[1]University of California Irvine
[2]Daegu Gyeongbuk Institute of Science and Technology
[3]University of California San Diego
{yni3, m.imani}@uci.edu, yeseongkim@dgist.ac.kr, tajana@ucsd.edu

Abstract

In this paper, we propose an efficient framework to accelerate a lightweight brain-inspired learning solution, hyperdimensional computing (HDC), on existing edge systems. Through algorithm-hardware co-design, we optimize the HDC models to run them on the low-power host CPU and machine learning accelerators like Edge TPU. By treating the lightweight HDC learning model as a hyper-wide neural network, we exploit the capabilities of the accelerator and machine learning platform, while reducing training runtime costs by using bootstrap aggregating. Our experimental results conducted on mobile CPU and the Edge TPU demonstrate that our framework achieves 4.5 times faster training and 4.2 times faster inference than the baseline platform. Furthermore, compared to the embedded ARM CPU, Raspberry Pi, with similar power consumption, our framework achieves 19.4 times faster training and 8.9 times faster inference.

1 Introduction

With the advent of the Internet of Things (IoT), numerous applications incorporate machine learning algorithms [Dall'Ora *et al.*, 2019]. However, the majority of these applications rely on highly complex algorithms, such as Deep Neural Networks (DNNs), which necessitate billions of parameters and extensive training time within a robust and high-performance computing environment [Pan and McElhannon, 2017; Li *et al.*, 2018; Shafique and others, 2020]. In light of the constrained memory and resources of edge devices, coupled with potential network disturbances and hardware failures inherent in IoT systems, current computing environments fall short of achieving real-time learning [Ram and others, 2021].

In contrast to existing machine learning, the human brain can train effortlessly and efficiently without much concern for noisy and broken neuron cells [Kanerva, 2009]. To more closely model the human brain, earlier researchers proposed HyperDimensional Computing (HDC) as an alternative computing method, which mimics important brain functionalities towards high-efficiency and noise-tolerant computation [Imani and others, 2017b; Hernández-Cano and others, 2021a]. HDC is motivated by the observation that the human brain operates on high-dimensional data representations. In HDC, objects and data are thereby encoded with high-dimensional vectors, called *hypervectors*, which have 10,000 or more elements, to perform learning tasks with computation in the high-dimensional space. HDC is well suited to address learning on edge systems as HDC models are computationally efficient to train [Imani and others, 2017a; Genssler and others, 2021; Halawani *et al.*, 2021], offer intuitive and human-interpretablity [Zou and others, 2021a], and provide strong robustness to noise [Poduval and others, 2021b; Imani and others, 2019].

These features make HDC a promising solution for today's embedded devices with limited storage, battery, and resources. To fully utilize its lightweight characteristics, most prior works rely on ASIC or emerging hardware acceleration. However, these designs are not commercially available and need a relatively long period to synthesize and fabricate after deriving the new applications.

In this paper, we show an acceleration solution for HDC by exploiting and reusing readily available and standardized DNN accelerators to ease the deployment of HDC in the real world, particularly focusing on Google Edge TPU. We propose a framework for efficient acceleration of the HDC in the edge environment by optimizing its algorithm to fully utilize the low-power Edge TPU as well as the host CPU. Our framework maps the HDC model to TensorFlow which works seamlessly with the Edge TPU. The bulky matrix operations in HDC efficiently utilize hardware parallelism. We further accelerate the training runtime on a low-power ARM CPU at the host by employing the ensemble methods based on bootstrap aggregating (bagging). Our results show that the joint experiments on mobile CPU and the Edge TPU show that our framework achieves 4.5× faster training and 4.2× faster inference than the baseline platform. In addition, our framework achieves 19.4× faster training and 8.9× faster inference compared to the embedded ARM CPU, Raspberry Pi, that consumes similar power consumption.

*The original paper was published in the proceedings of the 2022 Design, Automation and Test in Europe Conference (DATE). Yeseong Kim and Mohsen Imani are the co-corresponding authors of the paper.

2 Related Work

HDC, inspired by the large neural circuits inside the human brain, is both efficient in the calculation and robust against noise. Recent researchers have shown those advantages in multiple HDC applications, e.g., gesture/object detection [Moin and others, 2021; Zou and others, 2021b], DNA pattern matching [Kim and others, 2020; Poduval and others, 2021a], regression [Hernández-Cano and others, 2021b], and manufacturing [Chen and others, 2021], and clustering [Imani and others, 2020]. Recently, an increasing number of researchers focused on the acceleration of HDC utilizing the parallelism of its hardware-friendly operations. Multiple hardware platforms have been used to accelerate the training and inference process, e.g., FPGA [Schmuck and others, 2019] and ASIC [Datta and others, 2019; Khaleghi and others, 2020; Karunaratne and others, 2020; Wu and others, 2018]. However, their custom application-specific designs are less likely to be available due to the high manufacturing cost. In this paper, we tackle this issue by devising an HDC acceleration framework based on a readily available low-power platform, i.e., Google Edge TPU accelerator.

3 Hyperdimensional Computing at the Edge

The primary objective of this paper is to effectively map HDC operations for both training and inference to viable DNN-like models, thereby enabling acceleration through low-power Edge TPU. Figure 1 shows the overview of our framework. In the training phase, we generate base hypervectors randomly using the normal distribution. Then, it takes samples from the training dataset and encodes them on the Edge TPU. These encoded hypervectors are then sent to the host CPU to update the class hypervectors. During the inference phase, our framework relies on an inference neural network model parameterized with the trained class hypervectors and base hypervectors. The model is loaded onto the accelerator and operates on samples from the testing dataset. By effectively mapping the entire inference process to the Edge TPU, our framework enables real-time and efficient predictions.

3.1 Mapping HDC to Edge TPU

Inference in the HDC is performed with three major subtasks, input vectors encoding, class hypervectors update, and classification. We can view it as a three-layer wide neural network. The first part of the network includes the input layer with n nodes and the wide hidden layer with d nodes, and it maps the inputs to higher dimensions. The second part of the network takes the hidden layer as inputs and generates the classification results at the output layer with k nodes.

Encoding. As the basis of HDC, encoding maps the input space to higher dimensions, e.g., $d = 10,000$. Suppose an n-feature input sample vector has the form $\vec{F} = \{f_1, f_2, \ldots, f_n\}$. Following the non-linear encoding [Imani and others, 2020], our mapping relies on randomly generated $1 \times d$ base hypervectors $\{\overrightarrow{B}_1, \overrightarrow{B}_2, \ldots, \overrightarrow{B}_n\}$ for each input feature, and the randomness is realized through the normal distribution for components in these hypervectors, i.e., $b \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 1$. The components of these random hypervectors follow a symmetric distribution around zero so that the dot product between any two base hypervectors is very close to zero. Thus, we also regard them as near orthogonal. Then these hypervectors multiply with corresponding feature values, and the outputs are aggregated as $1 \times d$ encoded hypervectors:

$$\overrightarrow{E} = \tanh(f_1 \times \overrightarrow{B}_1 + f_2 \times \overrightarrow{B}_2 + \ldots + f_n \times \overrightarrow{B}_n)$$

We refer to this hypervector addition as the bundling operation, which preserves the information of each hypervector. Especially for non-linear encoding, we take the hyperbolic tangent value of each component in the encoded hypervector.

To map the encoding process to Edge TPU, we notice that bundling in the encoding process is essentially a large number of MAC operations. In other words, the encoding is indeed a vector-matrix multiplication that is ready to accelerate on most hardware accelerators. We map these MAC operations to the first half of that three-layer wide NN. Within that half, the input is the $1 \times n$ sample vector, and all the base hypervectors form the $n \times d$ weight matrix for edges connecting the input layer and the hidden layer. The non-linear term can be integrated into the NN as an activation function of nodes in the wide hidden layer.

Class hypervectors update. The process for generating and updating k class hypervectors uses lightweight operations called bundling. Starting with all zeros in the $1 \times d$ class hypervectors $\{\overrightarrow{C}_1, \overrightarrow{C}_2, \cdots, \overrightarrow{C}_k\}$, these hypervectors are updated according to the classification correctness of each input. For instance, if an encoded hypervector $\overrightarrow{E}_m$ that belongs to class a, is instead classified to b incorrectly, then the HDC training algorithm will update the class hypervectors for both class a and b with the bundling, $\overrightarrow{C}_a = \overrightarrow{C}_a + \lambda\overrightarrow{E}_m$ and $\overrightarrow{C}_b = \overrightarrow{C}_b - \lambda\overrightarrow{E}_m$. where λ is the learning rate.

Class hypervector training is interpreted as weights update in the neural network. Most edge accelerators are not designed for training or weights update. For example, Edge TPU lacks support for element-wise operations, so the acceleration for class hypervectors update is not available. Thus, we deploy this part on the host CPU due to this limitation; we devised an ensemble learning-based optimization method for higher efficiency as discussed in the next section.

Classification. In the final classification step of the HDC, the associative search checks the similarity between encoded query hypervectors and class hypervectors. We compute the similarity using the dot product: $\delta(\overrightarrow{E}_m, \overrightarrow{C}_k) = \overrightarrow{E}_m \cdot \overrightarrow{C}_k$. After repeating this for each class hypervector, the final result is the class with the highest similarity. Similarity check maps to the second half of that three-layer fully connected network. The input to this half is the encoded hypervectors coming out of the wide hidden layer, and the network parameters, i.e., the $d \times k$ weight matrix, are determined by the trained class hypervectors. Through the network, the output is the sum of the product between hidden node outputs and the weight on edge, which is the same calculation of the similarity check.

3.2 Boost Efficiency with Bagging

As mentioned in Section 3.1, a large portion of the training process is not accelerated due to the limitation of Edge TPU,

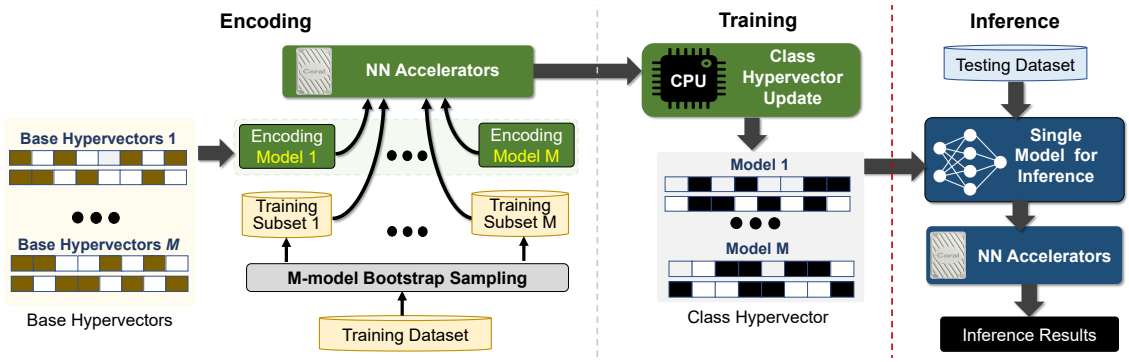

Figure 1: HDC acceleration framework with multi-model bootstrap sampling (Bagging) for runtime reduction

which has no support for on-device weights update. Low-power host machines at the edge also lack the computing power for fast training with wide hypervectors. This leads to our idea behind the use of the bagging method, which is to achieve more efficient training without compromising the HDC learning quality.

Bagging for faster HDC training. Bagging and other ensemble methods provide higher accuracy but often require the cost of longer training runtime because of the need for training multiple sub-models. However, we interpret its accuracy advantage from another angle. Because of the aggregation and the consensus-based prediction process, bagging is able to improve the accuracy of each sub-model. In other words, multiple smaller-sized models with fewer training iterations can still provide similar accuracy compared with fully-trained models. This outcome aligns with our targets, i.e., lower runtime cost of weights update. The cost of weights update is directly related to the class hypervector width. However, training multiple sub-models significantly increases the computation cost if the hypervector width stays unchanged. Thus, we decrease the hypervector width to $d' = d/M$, where d is the original hypervector width and M is the number of sub-models generated. Next, we further reduce the runtime through fewer training iterations. We utilize the property of bagging that it does not require fully-trained sub-models. The bagging method achieves similar accuracy with fewer than half iterations, which dramatically lowers the runtime. Then through the bootstrap sampling in bagging, we also speed up the training process. Both dataset and feature sampling generate a training subset, which means less computation cost.

Inference model generation. Even though we utilize the bagging method mainly for optimizing the training runtime on the host CPU, the inference process on Edge TPU also needs to adapt accordingly. As the bagging method trained multiple sub-models, which also take inputs from different sub-datasets, accelerating these models on Edge TPU in series is not efficient. Most Edge TPU only take one model at a time, and the weights have to be loaded to the on-chip buffer every time. This brings the overhead for preparing the accelerators for multiple models. Also, these sub-models trained through bagging are smaller, so running them in series may not fully utilize accelerator hardware parallelism. Thus, we design a technique to combine multiple trained sub-models as a single full-sized inference model.

For an n-feature input $\vec{F}$, it first passes through the encoding process, where its features are sampled and then different groups of base hypervectors encode it. Suppose the bagging process uses M different sub-models, and each of them is based on an $n \times d'$ matrix of base hypervectors $\boldsymbol{\mathcal{B}}^m = \{\overrightarrow{B}_1^m, \overrightarrow{B}_2^m, \cdots, \overrightarrow{B}_n^m\}$. For this matrix of each sub-model, some of the columns are set to zero, because they correspond to features that are not sampled. In this way, the feature sampling process is automatically finished. Then we stack these matrices horizontally to form a full $n \times d$ weight matrix $\mathbb{B}$ for the encoding part of the inference model. The output $1 \times d$ encoded hypervector $\vec{\mathbf{E}}$ is calculated as below:

$$\overrightarrow{E} = \vec{F} \times \mathbb{B} = \vec{F} \times [\boldsymbol{\mathcal{B}}^1 \, \boldsymbol{\mathcal{B}}^2 \, \ldots \, \boldsymbol{\mathcal{B}}^M]$$

Then these encoded hypervectors enter the second half of the inference model for classification of k different classes. For the different groups of class hypervectors trained through the bagging, we also stack them vertically to form a bigger $d \times k$ matrix $\mathbb{C}$ with the same dimension as before:

$$\vec{\mathbf{O}} = \overrightarrow{E} \times \mathbb{C} = \overrightarrow{E} \times [\boldsymbol{\mathcal{C}}^1 \, \boldsymbol{\mathcal{C}}^2 \, \cdots \, \boldsymbol{\mathcal{C}}^M]^T$$

where the $d' \times k$ matrix $\boldsymbol{\mathcal{C}}^m = \{\overrightarrow{C}_1^m, \overrightarrow{C}_2^m, \cdots, \overrightarrow{C}_k^m\}$. Instead of aggregating the results of similarity checks for each sub-model, i.e., multiple vector-matrix multiplications followed by an element-wise addition, we could perform a single time vector-matrix multiplication and the classification result is available directly using the aforementioned similarity search. Because the full inference NN model generated here has the same dimensions as the one generated without using bagging, it does not incur extra overhead during inference.

4 Experimental Results

We implement our framework on the Google Edge TPU connected through USB 3.0 to a lower-end laptop having a mobile Intel CPU i5-5250U. For the CPU baseline experiments and those on TPU without bagging, we train the model for 20 iterations to achieve fully trained models. For experiments on TPU with bagging, we trained 4 sub-models with hypervector width $d = 2500$ for 6 iterations. We used 60% of the training dataset for each sub-model. We tested five datasets

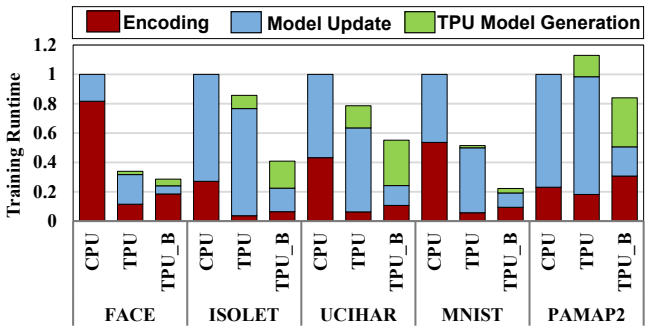

Figure 2: Training runtime cost comparison of three different framework settings: CPU baseline, TPU (without bagging method) and TPU_B (with bagging method). The runtime costs are normalized to the measurement on CPU within each dataset.

widely used in the HDC community: FACE [Kim and others, 2017], ISOLET [Cole and Fanty, 1994], UCIHAR [Anguita and others, 2013], MNIST [LeCun and others, 1998], and PAMAP2 [Reiss and others, 2012].

4.1 Training Efficiency

Figure 2 shows the runtime measurement for each component of the training process in HDC. Our framework without bagging, i.e., the TPU baseline, maps the training set encoding from the host CPU to Edge TPU. Compared with the CPU baseline, we observe that the encoding time, with the acceleration of Edge TPU, significantly decreases for four datasets. One exception is observed for PAMAP2 since the number of features in this dataset is relatively small, i.e., $n = 27$, being less compute-intensive during the encoding. Our framework is able to provide high encoding runtime speedup for datasets with large inputs. The maximum encoding runtime speedup is $9.37\times$ for the MNIST dataset. For datasets such as FACE, the encoding runtime takes up a large portion of the total training time, and our framework with the Edge TPU significantly reduces the overall runtime with $2.95\times$ speedup.

Our framework with bagging, i.e., TPU_B, achieves dramatically lower runtime for class hypervector update on the host CPU. For example, compared to the CPU baseline, the bagging method brings up to $4.74\times$ speedup for the hypervector update process in the host CPU. Compared with the overall training runtime in the CPU baseline, our framework brings up to $4.49\times$ speedup on the MNIST dataset by optimizing operations on both the Edge TPU and host CPU. It also achieves significantly faster training on FACE, ISOLET, and UCIHAR datasets with $3.49\times$, $2.45\times$ and $1.81\times$ speedup, respectively.

4.2 Inference Efficiency and Accuracy

Figure 3 shows the runtime measurements for the inference process of HDC on the Edge TPU. Besides our counterexample PAMAP2, our framework significantly accelerates the inference process. For example, the maximum inference speedup achieved on MNIST with the bagging method is $4.19\times$. The speedups for other 3 datasets are: $3.16\times$(FACE), $2.13\times$(ISOLET), $3.08\times$(UCIHAR). The bagging method, with a unified inference model, can achieve the inference runtime with no extra overhead compared with the TPU baseline.

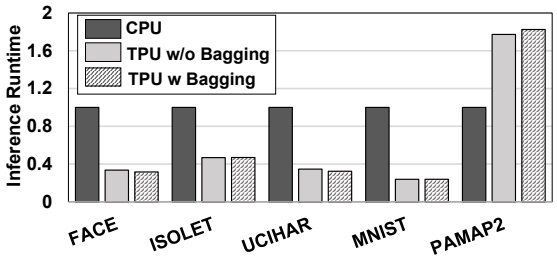

Figure 3: Inference runtime cost comparison: the runtime costs are normalized to the measurement on CPU within each dataset.

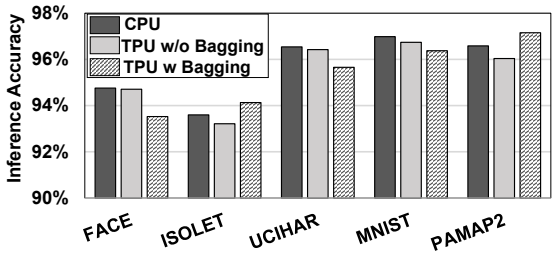

Figure 4: Inference accuracy for different framework settings

	FACE	ISOLET	UCIHAR	MNIST	PAMAP2
Training	21.5 ×	15.6×	17.9×	23.6×	18.6×
Inference	11.4×	7.2×	7.9×	11.1×	6.8×

Table 1: Our Edge TPU-based efficiency vs. Raspberry Pi 3.

Figure 4 summarizes the accuracy results of our experiments on five different datasets. The comparison illustrates that our proposed method is able to achieve similar inference accuracy on the Edge TPU. Because the ensemble method compensates for the possible incorrect classification of each sub-model, the final inference model even achieves higher accuracy than the full-sized, fully-trained model for some datasets, e.g., ISOLET and PAMAP2.

We also compare our Edge TPU-based platform with an embedded CPU which consumes similar average power consumption. Table 1 shows the performance improvement of our framework as compared to Raspberry Pi 3 using ARM Cortex A53 processor. Our results indicate that our framework can provide $19.4\times$ and $8.9\times$ faster training and inference compared to Raspberry Pi 3.

5 Conclusion

We propose a framework for efficient acceleration of HDC in the edge environment which optimizes the algorithm to fully utilize the low-power Edge TPU and the host CPU. It interprets the major computation of HDC as a hyper-wide lightweight NN and further accelerates the training on the host CPU by employing bagging. Our evaluation showed that the proposed framework offers significant efficiency improvements in training and inference on edge platforms.

Acknowledgments

This work was supported in part by National Science Foundation (NSF) #2127780, #2112167, #2052809, #2003279 and Semiconductor Research Corporation (SRC) Task #2988.001, SRC CRISP, and Department of the Navy, Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, and a generous gift from Cisco. It was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science and ICT) (NRF-2018R1A5A1060031).

References

[Anguita and others, 2013] Davide Anguita et al. A public domain dataset for human activity recognition using smartphones. In *ESANN*, 2013.

[Chen and others, 2021] Ruimin Chen et al. Joint active search and neuromorphic computing for efficient data exploitation and monitoring in additive manufacturing. *Journal of Manufacturing Processes*, 71:743–752, 2021.

[Cole and Fanty, 1994] Ron Cole and Mark Fanty. ISOLET. UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C51G69.

[Dall'Ora *et al.*, 2019] Nicola Dall'Ora, Stefano Centomo, and Franco Fummi. Industrial-iot data analysis exploiting electronic design automation techniques. In *IWASI*, pages 103–109. IEEE, 2019.

[Datta and others, 2019] Sohum Datta et al. A programmable hyper-dimensional processor architecture for human-centric iot. *JETCAS*, 9(3):439–452, 2019.

[Genssler and others, 2021] Paul R Genssler et al. Brain-inspired computing for wafer map defect pattern classification. In *ITC*, pages 123–132. IEEE, 2021.

[Halawani *et al.*, 2021] Yasmin Halawani, Dima Kilani, Eman Hassan, Huruy Tesfai, Hani Saleh, and Baker Mohammad. Rram-based cam combined with time-domain circuits for hyperdimensional computing. *Scientific reports*, 11(1):19848, 2021.

[Hernández-Cano and others, 2021a] Alejandro Hernández-Cano et al. Real-time and robust hyperdimensional classification. In *GLSVLSI*, pages 397–402, 2021.

[Hernández-Cano and others, 2021b] Alejandro Hernández-Cano et al. Reghd: Robust and efficient regression in hyper-dimensional learning system. In *DAC*, pages 7–12. IEEE, 2021.

[Imani and others, 2017a] Mohsen Imani et al. Exploring hyperdimensional associative memory. In *HPCA*, pages 445–456. IEEE, 2017.

[Imani and others, 2017b] Mohsen Imani et al. Voicehd: Hyperdimensional computing for efficient speech recognition. In *ICRC*, pages 1–8. IEEE, 2017.

[Imani and others, 2019] Mohsen Imani et al. A framework for collaborative learning in secure high-dimensional space. In *CLOUD*, pages 435–446. IEEE, 2019.

[Imani and others, 2020] Mohsen Imani et al. Dual: Acceleration of clustering algorithms using digital-based processing in-memory. In *MICRO*, pages 356–371. IEEE, 2020.

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

[Karunaratne and others, 2020] Geethan Karunaratne et al. In-memory hyperdimensional computing. *Nature Electronics*, 3(6):327–337, 2020.

[Khaleghi and others, 2020] Behnam Khaleghi et al. Shear er: highly-efficient hyperdimensional computing by software-hardware enabled multifold approximation. In *ISLPED*, pages 241–246, 2020.

[Kim and others, 2017] Yeseong Kim et al. ORCHARD: visual object recognition accelerator based on approximate in-memory processing. In *ICCAD*, pages 25–32. IEEE, 2017.

[Kim and others, 2020] Yeseong Kim et al. Geniehd: Efficient DNA pattern matching accelerator using hyperdimensional computing. In *DATE*, pages 115–120. IEEE, 2020.

[LeCun and others, 1998] Yann LeCun et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Li *et al.*, 2018] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.

[Moin and others, 2021] Ali Moin et al. A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition. *Nature Electronics*, 4(1), 2021.

[Pan and McElhannon, 2017] Jianli Pan and James McElhannon. Future edge cloud and edge computing for internet of things applications. *IEEE IOT*, 5(1):439–449, 2017.

[Poduval and others, 2021a] Prathyush Poduval et al. Cognitive correlative encoding for genome sequence matching in hyperdimensional system. In *DAC*, pages 781–786. IEEE, 2021.

[Poduval and others, 2021b] Prathyush Poduval et al. Stochd: Stochastic hyperdimensional system for efficient and robust learning from raw data. In *DAC*, pages 1195–1200. IEEE, 2021.

[Ram and others, 2021] Saswat Kumar Ram et al. Sehs: Solar energy harvesting system for iot edge node devices. In *Progress in Advanced Computing and Intelligent Engineering*, pages 432–443. Springer, 2021.

[Reiss and others, 2012] Attila Reiss et al. Introducing a new benchmarked dataset for activity monitoring. In *ISWC*, pages 108–109. IEEE, 2012.

[Schmuck and others, 2019] Manuel Schmuck et al. Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory. *JETC*, 15(4):1–25, 2019.

[Shafique and others, 2020] Muhammad Shafique et al. Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead. *IEEE Design & Test*, 2020.

[Wu and others, 2018] Tony F Wu et al. Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study. In *ISSCC*, pages 492–494. IEEE, 2018.

[Zou and others, 2021a] Zhuowen Zou et al. Scalable edge-based hyperdimensional learning system with brain-like neural adaptation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[Zou and others, 2021b] Zhuowen Zou et al. Spiking hyperdimensional network: Neuromorphic models integrated with memory-inspired framework. *arXiv preprint arXiv:2110.00214*, 2021.