2024 Design, Automation & Test in Europe Conference (DATE 2024)
Special Initiative "Autonomous Systems Design"

Frontiers in Edge AI with RISC-V: Hyperdimensional Computing vs. Quantized Neural Networks

Paul R. Genssler*, Sandy A. Wasif†, Miran Wael†, Rodion Novkin†, and Hussam Amrouch*†
*Semiconductor Test and Reliability, University of Stuttgart, Stuttgart, Germany
†Chair of AI Processor Design, Technical University of Munich; TUM School of Computation, Information and Technology; Munich Institute of Robotics and Machine Intelligence, Munich, Germany
Email: genssler@iti.uni-stuttgart.de; amrouch@tum.de

***Abstract*—Hyperdimensional Computing (HDC) is an emerging paradigm that stands as a compelling alternative to conventional Deep Learning algorithms. HDC holds four key promises. First, the ability to learn from little data. Second, to be robust against noise in this data. HDC also promises to be resilient against errors in the underlying hardware. This includes the memory on which the model is stored and errors in the computations of the operations, which is attributed to the encoding of information across an expansive dimensional space. Fourth, HDC can be implemented efficiently in hardware due to its lightweight and embarrassingly parallel computations. In this work, those four key promises are evaluated in a holistic way. A fixed-point and a binary HDC implementation are compared against neural network implementations. The models are executed on a RISC-V processor to ensure a fair comparison. While the results confirm the ability to learn from little data and the resiliency against errors, the higher inference accuracy of neural networks favors them in most experiments. Based on these insights, we formulate challenges and opportunities for HDC.**

Our implementations for QNN, binary and fixed-point HDC are available online: https://github.com/TUM-AIPro/HDC-vs-QNN

***Index Terms*—Hyperdimensional computing, Deep learning, Machine Learning, Edge AI, RISC-V, Reliability**

I. Introduction

Brain-inspired hyperdimensional computing (HDC) has emerged as an alternative machine learning concept. Large vectors are used to encode data and represent the patterns of this data. These vectors are typically in the dimension of thousands and thus referred to as hypervectors. Through the encoding, two similar data samples will have two similar hypervectors enabling classification, clustering [1], or reinforcement learning [2]. HDC has been applied in many domains, including semiconductor test [3] and reliability [4], transistor aging [5], and others [6, 7]. The interest in this alternative machine learning concept stems from its promises to combine several key features important for edge AI system. In such systems, no large datasets are available to train an edge AI model. Hence, it has to be able to learn from little data to adapt to its deployment environment. Incoming data is of low quality because the available sensors have to be cheap. The underlying hardware executing the machine learning model is exposed to harsh environmental conditions and power-constraint which can lead to errors in the computations. Cost constraints during manufacturing can result in unreliable data memories. HDC is positioned to address all of these concerns.

This work was partially supported by the German Federal Ministry of Education and Research and the German Academic Exchange Service (“Green AI” Project). P. Genssler’s work on HDC was supported by Advantest as part of the Graduate School “Intelligent Methods for Test and Reliability” at the Uni. of Stuttgart.

(1) One promise of HDC is the ability to **learn from little data**. In some domains, e.g., seizure detection by monitoring EEG signals, a single or a few samples are sufficient [8]. This ability allows for a quick model creation in scenarios where labeled samples are either rare, e.g., seizures, or expensive to create, e.g., prototype chips of a new transistor technology. The HDC model is not randomly initialized and then trained to adapted to the data. Instead, the HDC model is build directly from the data and thus a single samples can be sufficient. Diverse samples from the same class are still required to capture their distinct features accurately. In addition, learning from little data reduces the energy required to create the model. In combination with the reduced time for learning, HDC is a promising solution for edge AI devices for fast and energy-efficient learning.

(2) The second promise is the **robustness against noise** in the data samples. Each samples is not directly compared against the trained model but first encoded into the hyperspace. During this encoding, the low-dimensional data sample is mapped into a high-dimensional space. The resulting hypervector, while still impacted by the noise, often remain similar to a hypervector encoded from a noise-free sample. Through the large distances in the high-dimensional space, the impact of the noise is reduced.

(3) HDC models further exhibit a remarkable **resiliency against errors** in the underlying hardware. Computations can be incorrect, yet the model maintains its inference accuracy For example, the similarities between hypervectors do not need to be computed exactly as long as their relative similarly remains This enables very efficient implementations based on in-memory computing. The values of the hypervectors components are mapped to, eg, a polarization of a memory devices and the values of the unknown query hypervector are applied as voltages The resulting conductivity represents the similarity While extremely efficient, these analog computations are susceptible to voltage and temperature fluctuations and unreliable because of process variation in the underlying memory devices Although those similarly computations are incorrect, the model remains its accuracy because, first, all computations are impacted, and second, the high dimensionality offers enough redundancy The same reasoning applies to memories that store the values for traditional computing systems. If a bit flip occurs and changes the value of one component, there are still hundreds and thousands of other values that remain correct and ensure a correct inference despite the errors.

979-8-3503-4859-0/DATE24/© 2024 EDAA

(4) The fourth promise is a **lightweight implementation** of the HDC operations. Instead of large matrix multiplications, like in deep neural networks (DNNs), operations with hypervectors are based on simple arithmetic computations, such as addition or XOR on binary values. The HDC algorithm supports different underlying realizations, e.g., hypervector components can be real numbers or simple bits. Especially binary HDC enables very efficient hardware solutions by avoiding costly floating-point multipliers. In addition, operations on hypervectors are embarrassingly parallel because each dimension is independent and can be processed separately.

Our novel contributions in this work are as follows. We revisit the four promises of HDC by comparing fixed-point and binary HDC models against conventional deep learning models, namely convolutional neural networks (CNNs) and DNNs, in a holistic way. First, we evaluate the inference accuracy after learning from little data, the impact of noise on the data samples, and memory errors in the models' storage. Then, the models are quantized, if applicable, and executed on an embedded edge AI system. Using a 32-bit RISC-V processor, we measure the number of cycles it takes for an inference and to train a model. We repeat the experiments for four datasets from different domains to capture a broader scope.

II. Hyperdimensional Computing

HDC is inspired by the human brain that processes a large amount of data and recognizes patterns, rather than absolute values, to reach decisions [9]. Following that idea, information must be mapped from their original confined representation into a high dimensional representation referred to as hyperspace. The hypervectors should be larger than the features existing in the original data to introduce redundancy in the representation.

A. HDC Algebra

Brain-inspired computing follows a different methodology than normal computing paradigms. It introduces its own algebra with arithmetic and logical operators. The hypervector components can be represented by binary, integer, real, or even complex data types. Operations in the HDC algebra are bundling, binding, and permutations [9]. Bundling joins multiple hypervectors into one and combines their patterns. It can be implemented as component-wise addition or majority vote for non-binary (e.g., floating or fixed point) or binary data types, respectively. Binding combines a key and a value hypervector into a single key-value pair hypervector. Binding is achieved by component-wise multiplication or XOR for non-binary or binary data types, respectively. Permutation shuffles the components inside one hypervector, e.g., through rotation or multiplication with a permutation matrix. Finally, the similarity between hypervectors is computed using one of two metrics, cosine similarity or Hamming distance for non-binary or binary.

B. HDC Encoding

The HDC classification process starts with the encoding, which maps data from the real-world domain into hyperspace. The encoding efficiently spreads the data and maps similar data points onto similar hypervectors. A common encoding technique for HDC is the record-based encoding, where each feature is represented as a key-value pair. Initially, a list of key and value hypervectors is created randomly and remains constant. For every feature, its key hypervector is bound with a value hypervector that is selected based on the quantized real-world feature value. All key-value hypervectors are then bundled into a single sample hypervector. Another encoding method is the Gaussian encoding to introduce non-linearity in the mapping process. A randomly generated matrix, following the standard normal distribution, is multiplied with the feature vector of the sample.

C. HDC Training and Inference

For classification applications, the HDC model is created by bundling sample hypervectors of each class into class hypervectors. During retraining, the inference is performed on the training data. For each wrong inference, the model is updated by adding the sample hypervector to the correct class and subtracting it from the wrong class. This increase the similarity with the correct class while reducing it with the incorrect one. During inference of the test data, the same encoding algorithm is applied.

D. HDC Implementation

Various HDC implementations have been proposed, ranging from software frameworks [10] to fully custom hardware [11, 12]. A suitable HDC implementation dependents on the application and the target platform. Binary HDC is more suitable for FPGAs or edge devices as a platform since it is characterized by large dimensionality and simple operations (e.g., XOR). Non-binary, e.g., floating-point representations, introduce more demanding operations (multiplication) but the full precision of the components allows a reduction of the hypervector dimension [13]. Hence, it is more suitable for GPUs as they are designed to operate with non-binary values and can easily support the required complex arithmetic operations.

III. Experimental Setup

All experiments are performed for four different datasets: Digit MNIST, Fashion MNIST, ISOLET and UCI-HAR. A binary and a fixed-point HDC model are implemented. To compare, three neural networks are created. All experiments are repeated ten times to reduce the impact of randomness.

A. Binary HDC Implementation

The encoding method depends on the dataset. For image classification, the pixels are binarized into black and white. Then, only key hypervectors of all white pixels are bundled. For the tabular datasets (ISOLET, UCI-HAR), the record-based encoding is utilized. To represent binary hypervector components efficiently in C code, 32 dimensions are packed into an integer. Bit-wise operations enable efficient binding and permutation. During the encoding, non-linear quantization is used to evenly split the data and generate the value hypervectors. The non-linearity allows to minimize the number of value hypervectors to four. The Hamming distance is used as the similarity metric for classification. Ten epochs of retraining are performed and the hypervectors have a dimension of 4096 bits.

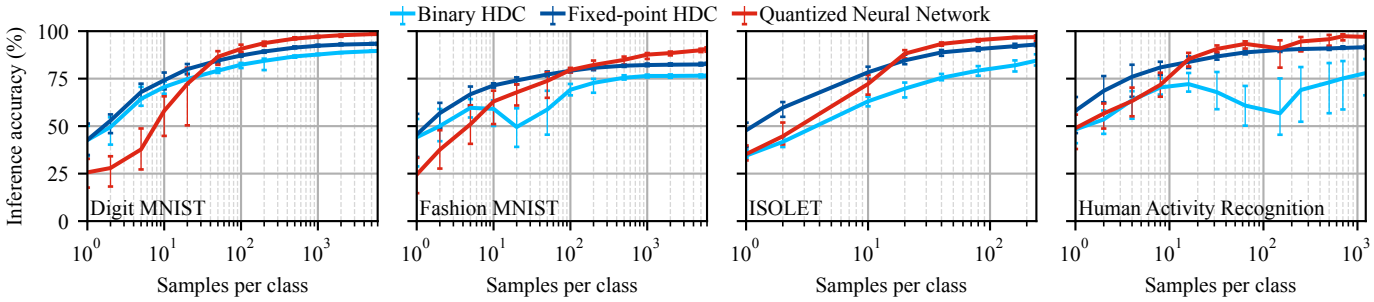

Figure 1. Hyperdimensional computing can learn from little data and achieve a higher inference accuracy than neural networks. However, the latter catch up quickly and ultimately outperform HDC-based models. The experiments are repeated ten times.

B. Fixed-Point HDC Implementation

The fixed-point HDC implementation is based on [13]. It is applied to all four datasets since the non-linear Gaussian encoding is not application specific and only depends on the number of features. However, it introduces a large matrix multiplications as part of the encoding. The model is created through single pass training, which initializes the model by bundling a portion of the training data and then performing retraining. Ten epochs of retraining are performed. The higher precision of the fixed-point model allows a reduction of the dimension to 2000.

During inference, the hypervector components are quantized to 16-bit fixed-point values. In the encoding, the sine/cosine functions are computationally costly for the large 32-bit value space. With the 16-bit fixed-point model, precomputed values are stored in a code book in an offline phase to significantly reduced the code complexity during the online phase.

C. Neural Network Implementations

The neural networks are implemented with the PyTorch framework [14]. Since Digit and Fashion MNIST datasets contain two-dimensional images, a simple CNN is constructed to achieve better results. The CNN consists of two Conv2d-BatchNorm-ReLU-MaxPool blocks, a Linear-BatchNorm-ReLU block, and an output linear layer. ISOLET and UCI-HAR datasets contain one-dimensional vector inputs with a relatively low number of features, thus a multilayer perceptron (MLP) is sufficient to learn the pattern. MLPs for both datasets have a similar design consisting of three Linear-BatchNorm-ReLU blocks followed by an output linear layer. All network architectures are designed manually to ensure high inference accuracy while keeping the size of the networks and number of calculations low.

Additionally, the weights and activations in all neural network implementations are quantized to eight bits using the PyTorch framework. A general mapping Q of floating point parameters w_i to 8-bit quantized ones can be represented as

$$Q(w_i) = min(-128, max(127, round(\frac{w_i}{S}) + Z))$$

where S is a scaling factor and Z is a zero-point. Quantization-aware training with per-channel quantization is employed, meaning that scaling factors and zero points are calculated for all elements along one of the dimensions of a tensor, which yields better accuracy compared to per-tensor quantization.

IV. Learning from Little Data

HDC offers a novel, alternative approach to creating a machine learning model. As described in Sec. II-C, HDC models are initialized with the encoded samples themselves. In other words, one encoded and labeled training sample already constitutes a model. Each additional hypervector either starts a new class or is bundled into the existing class hypervectors thereby increasing their coverage of the feature space.

The advantages of this approach can be seen in Fig. 1. Compared to the CNN for the MNIST datasets, an about 20 % higher inference accuracy can be achieved from training with a single sample per class. Even though these initial models only achieve a low accuracy between 40 % and 60 %, it shows that learning from little data is possible with HDC. However, the neural networks quickly catch up with HDC and at twenty samples per class do they match its performance. Only for the more complex Fashion MNIST dataset are one hundert samples per class required.

Binary HDC shows a mixed performance. In the image datasets, it has a jump start similar to fixed-point HDC but is matched at fewer samples because of its overall lower achievable inference accuracy. For the time-series datasets, binary HDC actually learns as fast as or slightly slower than the neural networks. Those MLPs do not have convolutional layers and thus more trainable parameters, yet they can achieve a high inference accuracy at fewer samples.

V. Robustness against Noise

The second promise is the robustness against noise in the data samples. The inherent redundancy and distributed nature of hyperdimensional representations promises to contribute to their ability to filter out noise and enhance the overall resilience of the learning process. In an embedded edge AI system, this noise could have many sources ranging from environmental conditions to the quality of the sensors. To evaluate the robustness against noise, noise is applied to the samples during the inference phase.

To allow for easier comparability to different sources of noise in the field, Gaussian noise is applied to the test samples. First, all features of a sample are normalized to a range of negative one to positive one. Then the Gaussian noise is applied with a μ of zero and sigma σ as the strength of the noise and the parameter to explore. A range from 0.01 to 1 is sufficient as shown in Fig. 2. All noisy values are clipped to the range of negative one to one and scaled back to their original value range.

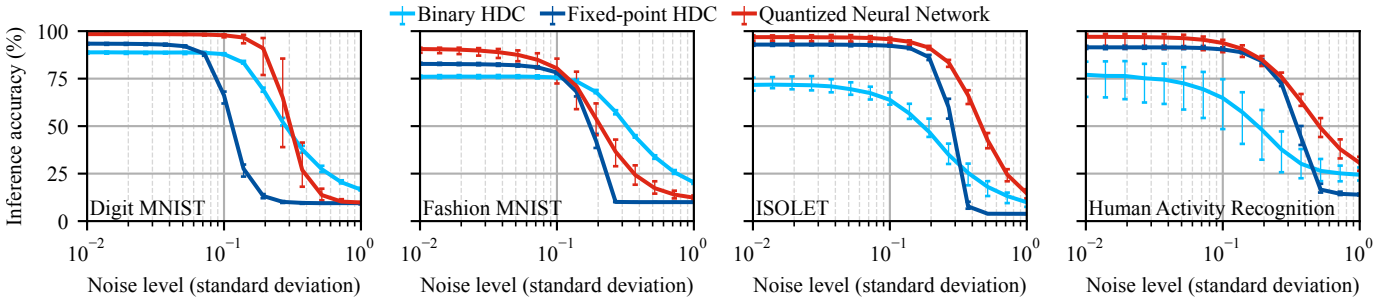

Figure 2. Noise injected in the test samples reduces the inference accuracy for all models. The higher baseline accuracy of the neural networks enables them to outperform HDC in most settings.

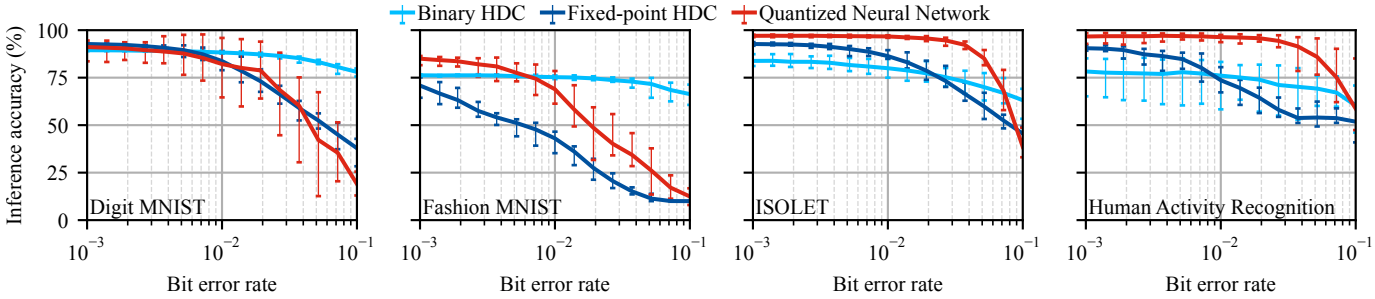

Figure 3. Bits in the models are flipped to simulate errors in the underlying hardware. Although binary HDC is the most robust, because of its low baseline accuracy, a high error rate is required for it to outperform the quantized neural works.

After training with the clean data, the noise is applied to the test dataset and the inference is performed.

The neural networks and the fixed-point HDC model behave alike. Initially, little noise has little impact until a threshold of around $\sigma = 0.1$ is reached, after which the inference accuracy starts to drop quickly. The fixed-point HDC model shows less robustness as it drops earlier. Although the binary HDC model behaves similar, it shows a more subtle decrease in inference accuracy. For the Fashion MNIST dataset, it even maintains its 75 % accuracy at a higher σ than the CNN and the fixed point HDC model. Nevertheless, overall there is no evidence for a higher robustness of HDC against noise in the data. The encoding maps the noise from the real-world values into the hyperspaces. The neural networks have the higher baseline accuracy, which helps them to maintain a higher accuracy under most noise levels.

VI. Resiliency against Errors

Embedded edge AI devices are deployed in great numbers, demanding a low cost in production and test. Therefore, the quality of the hardware can be lacking. Memories are an area where throughout testing becomes expensive and wear-out plays a major factor, especially for Flash and emerging memory technologies [15, 16]. However, not the memory itself and thus the whole system fails, but individual memory cells exhibit a variety of errors, such as stuck-at faults. If the machine learning model is stored in such a memory, individual bits will be corrupted and ultimately impact the inference accuracy.

A. Error Injection

For both HDC models, the errors are injected into the class hypervectors. In the binary model, those consist of simple bits, that are flipped and directly alter the class hypervector. In the fixed-point model, the 16-bit fixed point representations are converted into a binary string and the bit flips are applied. Afterwards, the 16-bit representation is restored. Similarly, in the neural networks, their weights are quantized into 8-bit integers in which the bit flips are injected. In both cases does the impact of a bit flip on the model only become clear after the conversion back into their number representation. The erroneous models are then directly used for inference, no further retraining or error corrections are performed. The initial training of the models is not error-aware.

B. Impact of Errors

The quantized neural networks for ISOLET and UCI-HAR are remarkably resilient as shown in Fig. 3. Only at a bit error rate of about 0.05 (i.e., five out of 100 bits are flipped) do the errors have an impact. The 8-bit quantized weights do not allow for an extremely large value to impact the overall network like it could do in floating point values. This limits the susceptibility. A similar behavior is seen in the fixed-point HDC model. However, the inference accuracy starts to drop at a significantly lower bit error rate, e.g., almost 10x for UCI-HAR. In combination with the lower baseline inference accuracy do the neural networks outperform fixed-point HDC for all benchmarks.

Interestingly, binary HDC is the most resilient. It shows the smallest drop in inference accuracy. In a binary hypervector, there is no most significant bit and thus no single bit flip has a measurable impact. Although it is the most resilient, binary HDC is the least accurate. Hence, its resiliency only provides a net benefit over the quantized neural networks for an error rate higher than 1×10^{-1}. Binary HDC outperforms fixed-point for even lower bit error rates, depending on the dataset.

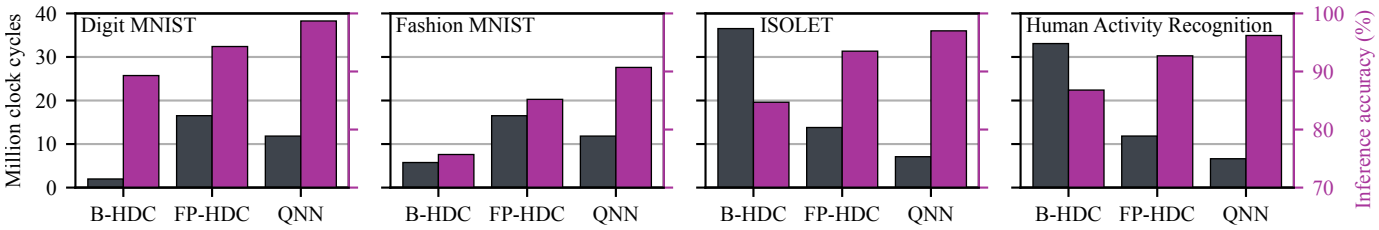

Figure 4. Number of clock cycles to perform one inference with binary HDC (B-HDC), fixed-point (FP-HDC), and the quantized CNN and MLPs (QNN).

In summary, binary HDC offers the promised high resiliency against errors in the underlying hardware. However, high error rates are required to compensate the low baseline inference accuracy and outperform quantized neural networks.

VII. Lightweight Implementation

In embedded edge AI systems, the inference accuracy is not the only deciding factor for a machine learning model. Other parameters such as execution time or power consumption play a vital role. Large models cannot fit onto the small memories and performing an inference would exceed the available time budget, e.g., recognize the change in activity and perform an action for the user.

A. Experimental Setup

The trained neural network models are converted into C-code using an in-house framework based on a TVM [17] compiler. First, a MicroTVM extension is employed to generate the models' C-library. Then, two header files are generated. One stores the samples and another the target classes based on the PyTorch model. Finally, the neural network generated by TVM is executed with the given input sample. The produced classification output is compared to the PyTorch model's output to confirm a correction functionality. The binary and fixed-point HDC models are written from scratch. To compare the three models, their execution is simulated on a small 32-bit RISC-V processor, which features a five-stage pipeline. This processor is offered by the Synopsys ASIP Designer [18]. The tool is used to compile the C-code of each model, perform the simulation, and measure the number of cycles. To this end, it generates the RTL code of the processor, which is then simulated in Synopsys VCS to extract a cycle-accurate run time. One hundred inferences are performed in one simulation and the average number of cycles is reported. The synthesis for power analysis is performed with the Synopsys 28/32nm educational library.

B. Run time, Energy, and Discussion

Fig. 4 visualizes the tradeoff between inference accuracy and the number of cycles needed to perform an inference. The run times vary depending on the dataset with the highest variation in the binary HDC model. For ISOLET and UCI-HAR, a quantization, binding, and bundling operations are necessary, where the latter requires a counter for each of the 4096 dimensions and represents a large cost factor. The fixed-point HDC model scales with the number of features, since they determine the size of the encoding matrix. Similarly, the CNN for the image datasets have a higher computational complexity than the MLPs. Overall, the quantized neural networks offer a higher accuracy than HDC but require sometimes more cycles to execute compared to binary HDC, depending on the dataset.

Power consumption draws a similar picture. Binary HDC consumes the least dynamic power, followed by fixed-point HDC and the CNN with a 1.36x and 1.45x higher consumption. However, due to the fixed-point's high runtime, its energy consumption per inference is the highest with 11.1x that of binary HDC. The CNN's consumption is 7.8x higher.

Next to the lower power consumption, one main advantage of HDC is its simple training procedure. In fact, the encoding consumes the most cycles but has to be only done once. Then, the sample hypervectors can be reused to perform retraining. To train a model from one hundred samples for ten epochs, cycles times increase by 1.3x-1.9x and by 1.0x-1.3x for fixed-point and binary, respectively. Numbers for the neural networks are not available as the TVM framework does not support model training. Nevertheless, they are at least 10x higher for ten training epochs because each epochs contains an inference pass. In summary, HDC enables model training on the edge by trading off inference accuracy with run time complexity and power consumption.

C. Memory Footprint of the Models

Besides run time, the memory footprint is an important cost and power factor for edge AI systems. The storage required to execute the model incurs cost in manufacturing, increases power consumption, and impacts the run time [19]. Fast and efficient implementations based on in-memory computing have been explored [20], but they can reduce the inference accuracy and do not consider the costly encoding step [4, 21, 22]. Thus, the focus of the analysis is on the general memory footprint of the models. HDC models require storage for the class hypervectors and for the encoding. Depending on the selected encoding method, number of features, and dimension, HDC models can become larger than MLPs. For example, the fixed-point HDC model for HAR and ISOLET occupies 2268 kB and 2572 kB, respectively. In contrast, the MLPs occupy 893 kB and 953 kB. The CNN is much more efficient and only occupies 37 kB, which is 11x of the binary HDC model. In summary, HDC is not a more memory-efficient algorithm than CNNs.

VIII. Findings, Conclusion, and Challenges

In this work, the promises of HDC were revisited and evaluated in an holistic way. A fixed-point and binary HDC model were compared against neural networks.

A. Key Findings on HDC Promises

(1) **Learning from little data** is possible and provides HDC a 10 % to 25 % boost in inference accuracy when training with one sample per class. This advantages fades already at 20 training samples per class for ISOLET and HAR whereas the CNN requires 100 samples to match the fixed-point HDC inference accuracy. Binary HDC does not show this advantage for ISOLET and HAR datasets.

(2) **Robustness against noise** in the data samples do all tested algorithms show to a similar degree. The neural networks and fixed-point HDC show a steeper drop in accuracy than binary HDC. Due to the lower initial inference accuracy of the HDC models, the neural networks provide the highest accuracy in all but the image datasets with high noise levels.

(3) **Resiliency against errors** is evident for the binary HDC model. It shows the lowest relative drop in inference accuracy. The fixed-point HDC model again behaves similar to and partially worse than the neural networks. However, the MLPs higher baseline accuracies are sufficient to still outperform binary HDC at high error rates. The CNN is less robust and outperformed by binary HDC.

(4) **Lightweight implementation** can be crafted for binary HDC as well as the neural networks, depending on the dataset. Simple bundle encodings use for MNIST are lightweight while the more complex record-based encoding for ISOLET and HAR takes more cycles to process than an MLP. Fixed-point HDC is consistently slower than the neural networks, has a higher memory footprint, and consumes more energy per inference operation. Yet, the CNN consumes 7.8x more energy than binary HDC for MNIST while offering a higher inference accuracy.

B. Perspective and Challenges

The key findings on the promises of HDC demonstrate the challenges HDC is facing. Its inference accuracy is not high enough to compete with neural networks. While other properties, such as robustness or runtime, are equally important in edge AI systems, HDC does not clearly outperform neural networks in robustness, resiliency, or runtime. The higher baseline inference accuracy of the neural networks compensates for the lower robustness or resiliency.

Current research on HDC for embedded edge AI systems mainly focuses on the inference phase. However, the advantages of HDC in this area are not clear, if inference accuracy is a concern. Instead, focusing on the training of the model is more promising. The results show that the computational complexity of the training is not significantly higher with HDC than for inference. This is due to the high cost of encoding, after which model adjustments are comparatively cheap. Therefore, the promise of HDC lies in the training on the edge.

ACKNOWLEDGMENT

We would like to thank P. Verbist and W. Geurts from Synopsys and their team for the valuable support. We thank the German University in Cairo (GUC) as well as Mohamed Abdel-Ghany, Eman Azab, and Maggie Mashaly from GUC for the insightful discussions and support to S. Wasif and M. Wael.

REFERENCES

[1] M. Imani, Y. Kim, et al., "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019, pp. 1591–1594.

[2] Y. Ni, M. Issa, et al., "Hdpg: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1141–1146.

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

[4] P. R. Genssler and H. Amrouch, "Brain-inspired computing for circuit reliability characterization," IEEE Transactions on Computers, vol. 71, no. 12, pp. 3336–3348, 2022.

[5] P. R. Genssler, H. E. Barkam, et al., "Modeling and predicting transistor aging under workload dependency using machine learning," IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I), 2023.

[6] P. R. Genssler, L. Alrahis, et al., "Hdcircuit: Brain-inspired hyperdimensional computing for circuit recognition," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE'24)*, 2024.

[7] H. Amrouch, F. Klemme, et al., "Design close to the edge in advanced technology using machine learning and brain-inspired algorithms," in *27th Asia and South Pacific Design Automation Conference*, 2022.

[8] A. Burrello, K. Schindler, et al., "One-shot learning for ieeg seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing," in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2018, pp. 1–4.

[9] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," Cognitive computation, vol. 1, pp. 139–159, 2009.

[10] M. Heddes, I. Nunes, et al., "Torchhd: An open source python library to support research on hyperdimensional computing and vector symbolic architectures," Journal of Machine Learning Research, vol. 24, no. 255, pp. 1–10, 2023.

[11] T. Zhang, S. Salamat, et al., "Hd2fpga: Automated framework for accelerating hyperdimensional computing on fpgas," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, IEEE, 2023, pp. 1–9.

[12] S. Salamat, M. Imani, et al., "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 53–62.

[13] A. Hernández-Cano, N. Matsumoto, et al., "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2021, pp. 56–61.

[14] A. Paszke, S. Gross, et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

[15] C. Hakert, K.-H. Chen, et al., "Software Based Read and Write Wear-Leveling for Non-Volatile Main Memory," ACM Transactions on Embedded Computing Systems (TECS), 2021.

[16] P. R. Genssler, V. Van Santen, et al., "On the reliability of fefet on-chip memory," IEEE Transactions on Computers, vol. 71, no. 4, pp. 947–958, 2022.

[17] T. Chen, T. Moreau, et al., "Tvm: An automated end-to-end optimizing compiler for deep learning," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18, Carlsbad, CA, USA: USENIX Association, 2018, pp. 579–594.

[18] Synopsys. "Asip designer." (2023), [Online]. Available: https://www.synopsys.com/dw/ipdir.php?ds=asip-designer.

[19] P. R. Genssler, A. Vas, et al., "Brain-inspired hyperdimensional computing: How thermal-friendly for edge computing?" IEEE Embedded Systems Letters, vol. 15, pp. 29–32, 2022.

[20] P. R. Genssler, M. Mayahinia, et al., "Drophd: Technology/algorithm co-design for reliable energy-efficient nvm-based hyperdimensional computing under voltage scaling," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, 2024.

[21] S. Kumar, S. Chatterjee, et al., "Cross-layer FeFET reliability modeling towards robust hyperdimensional computing," in *IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022.

[22] S. Thomann, H. L. G. Nguyen, et al., "All-in-memory brain-inspired computing using FeFET synapses," Frontiers in Electronics, vol. 3, 2022.