HyDREA: **Utilizing Hyperdimensional Computing for a More Robust and Efficient Machine Learning System**

JUSTIN MORRIS, University of California San Diego, La Jolla, CA and San Diego State University, San Diego, CA
KAZIM ERGUN and BEHNAM KHALEGHI, University of California San Diego, La Jolla, CA
MOHEN IMANI, University of California Irvine, Irvine, CA
BARIS AKSANLI, San Diego State University, San Diego, CA
TAJANA SIMUNIC, University of California San Diego, La Jolla, CA

Today's systems rely on sending all the data to the cloud and then using complex algorithms, such as Deep Neural Networks, which require billions of parameters and many hours to train a model. In contrast, the human brain can do much of this learning effortlessly. Hyperdimensional (HD) Computing aims to mimic the behavior of the human brain by utilizing high-dimensional representations. This leads to various desirable properties that other Machine Learning (ML) algorithms lack, such as robustness to noise in the system and simple, highly parallel operations. In this article, we propose HyDREA, a HyperDimensional Computing system that is Robust, Efficient, and Accurate. We propose a Processing-in-Memory (PIM) architecture that works in a federated learning environment with challenging communication scenarios that cause errors in the transmitted data. HyDREA adaptively changes the bitwidth of the model based on the signal-to-noise ratio (SNR) of the incoming sample to maintain the accuracy of the HD model while achieving significant speedup and energy efficiency. Our PIM architecture is able to achieve a speedup of 28× and 255× better energy efficiency compared to the baseline PIM architecture for Classification and achieves 32× speed up and 289× higher energy efficiency than the baseline architecture for Clustering. HyDREA is able to achieve this by relaxing hardware parameters to gain energy efficiency and speedup while introducing computational errors. We show experimentally, HD Computing is able to handle the errors without a significant drop in accuracy due to its unique robustness property. For wireless noise, we found that HyDREA is 48× more robust to noise than other comparable ML algorithms. Our results indicate that our proposed system loses less than 1% Classification accuracy, even in scenarios with an SNR of 6.64. We additionally test the robustness of using HD Computing for Clustering applications and found that our proposed system also looses less than 1% in the mutual information score, even in scenarios with an SNR under 7 dB, which is 57× more robust to noise than K-means.

CCS Concepts: • **Computing methodologies** → **Machine learning algorithms**; • **Hardware** → *Fault tolerance;*

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, in part by SRC-Global Research Collaboration Grant Task No. 2988.001, and also NSF Grants No. 1527034, No. 1730158, No. 1826967, No. 1911095, and No. 2003277.
Authors' addresses: J. Morris, University of California San Diego, La Jolla, CA 92093, USA, and San Diego State University, San Diego, CA 92182, USA; email: justinmorris@ucsd.edu; K. Ergun, B. Khaleghi, and T. Rosing, University of California San Diego, La Jolla, CA 92093, USA; emails: kergun@eng.ucsd.edu, bkhalegh@eng.ucsd.edu, tajana@ucsd.edu; M. Imani, University of California Irvine, Irvine, CA 92697, USA; email: m.imani@uci.edu; B. Aksanli, San Diego State University, San Diego, CA 92182, USA; email: baksanli@sdsu.edu.

1539-9087/2022/10-ART78
https://doi.org/10.1145/3524067

Additional Key Words and Phrases: Machine Learning, brain-insipred hyperdimensional computing, processing-in-memory

ACM Reference format:
Justin Morris, Kazim Ergun, Behnam Khaleghi, Mohen Imani, Baris Aksanli, and Tajana Simunic. 2022. HyDREA: Utilizing Hyperdimensional Computing for a More Robust and Efficient Machine Learning System. *ACM Trans. Embedd. Comput. Syst.* 21, 6, Article 78 (October 2022), 25 pages.
https://doi.org/10.1145/3524067

1 INTRODUCTION

"Federated learning" [1] is a popular model for distributed model training in which a centralized model stored on a server is "cloned" to some set of devices that all collect the same features. Each device then updates its local copy of the model and periodically transmits weights to the server, which are used to update the global model via an averaging operation. Intuitively, federated learning reduces communication costs by transmitting only model weights instead of raw training data.

In "Federated learning," **Hyperdimensional (HD)** computing offers three benefits [2]. First, an HD "model" is simply a collection of bitvectors, which may be less burdensome for communication than other state-of-the-art methods (especially deep neural networks) where the weights are typically floating point values and are non-negligible in size [3, 4]. While a line of deep neural networks research tries to reduce the parameters of these models [5], the number of parameters are still higher than HD. Second, local training of the HD model is extremely simple and more energy efficient than many existing ML techniques [6]. Third, transmitting faulty model weights in classical ML algorithms may lead to slower training or convergence to a worse local optimum compared to HD.

The third point is particularly helpful for "Federated learning." Transmitting model parameters to the central learning system is done mostly through wireless communication. The noise in a wireless channel can incur bit-level errors in the transmitted signal and without error correction, could lead to faulty models due to the noisy data. This is especially true in urban areas where distance is not the only factor adding noise to the wireless channel but also large buildings and multiple obstacles in the way that degrade the wireless signal.

We additionally take advantage of the simple and highly parallelizable operations in HD to create an analog PIM accelerator with adaptable model bitwidths to achieve the best energy and execution time, while maintaining high accuracy based on the SNR of the wireless channel. This characteristic has made HD the target of various hardware acceleration frameworks, particularly FPGAs [7], and PIM architectures [6, 8, 9]. Although GPUs and FPGAs provide a suitable degree of parallelism that makes them amenable to machine learning algorithms such as deep neural network [10], the complexity of their resources, e.g., floating point units or DSP blocks, is far beyond the HD requirements, making such devices inefficient for HD. Analog PIM architectures tackle this problem as they comprise memresistive arrays with intrinsically non-complex computational capability, which is sufficient for HD operations. Besides block-level parallelism, another remarkable feature of PIM is eliminating the high cost data movement in the traditional von Neumann architectures as, in PIM, data resides where computation is performed. Adding a PIM accelerator for HD computing to perform cognitive tasks provides significant speed up over utilizing the on-board CPU and saves energy with analog computations and less data movement. Our contributions in this article are as follows:

- We propose a PIM architecture that adaptively changes the bitwidth of the model based on the SNR of the incoming sample to maintain the accuracy of the HD model while achieving high speedup and energy efficiency. Our PIM architecture is able to achieve 255×

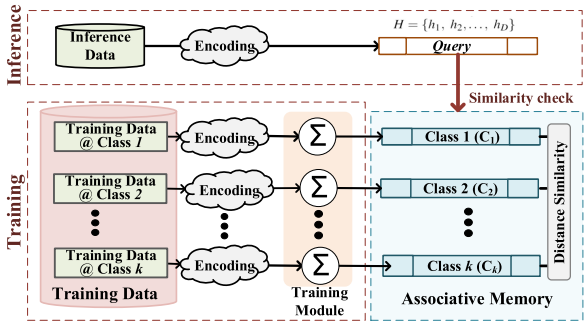


Fig. 1. Overview of HD model training and inference.

better energy efficiency and speed up execution time by 28× compared to the baseline PIM architecture.

- We take advantage of HD Computing's robustness to errors and relax the precision of ADCs in ISAAC [11], which introduces errors, but improves area and energy efficiency. Our architecture also utilizes quantized values to different bitwidths.
- We additionally evaluate utilizing our accelerator in a federated learning environment, by utilizing a popular network simulator—ns-3 [12]—to model the communication between devices and simulate wireless noise. We compared HyDREA with other light-weight ML algorithms in the same noisy environment. Our results demonstrate that HyDREA is 48× more robust to noise than other comparable ML algorithms. Our results indicate that our proposed system loses less than 1% Classification accuracy, even in scenarios with an SNR under 7 dB.
- We additionally evaluate HD Clustering to the same wireless communication errors and found that our proposed system also looses less than 1% in the mutual information score, even in scenarios with an SNR under 7 dB, which is 57× more robust to noise than K-means.
- Finally, we extend our architecture to support HD Clustering and our results show that our PIM architecture achieves 289× higher energy efficiency and 32× speed up compared to the baseline architecture during Clustering.

2 PRELIMINARY

In this section, we first explain the procedures involved in HD algorithm and then review the related work on HD acceleration and HD robustness to noise.

2.1 Hyperdimensional Computing Classification

Without loss of generality, we explain the steps of HD computing for Classification tasks, though other algorithms, e.g., Clustering, follow the same procedure, as well. These steps are illustrated by Figure 1.

(1) Encoding: There are multiple different types of encoding for HD Computing [13–15]. In this article, we evaluate two different types. The first is Random Projection and the second is ID-Level. Let us assume a feature vector $\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$, with n features ($f_i \in \mathbb{N}$) in original domain. The goal of the encoding stage is to map this feature vector to a D-dimensional space vector: $\mathbf{H} = \{h_1, h_2, \ldots, h_D\}$.

Random Projection: This encoding was first proposed in Reference [14]. This encoding first generates D dense bipolar vectors with the same dimensionality as original domain, $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_D\}$, where $\mathbf{p}_i \in \{-1, 1\}^n$. The inner product of a feature vector with each randomly generated vector gives us a single dimension of a hypervector in high-dimensional space. For

example, we can compute the ith dimension of the encoded data as

$$h_i = sign(\mathbf{p}_i \cdot \mathbf{F}),$$

where *sign* is a sign function that maps the result of the dot product to +1 or −1. Thus, to encode a feature vector into a hypervector, we perform a matrix vector multiplication between the projection matrix and the feature vector using:

$$\mathbf{H} = sign(\mathbf{PF})$$

ID-Level: This encoding was first proposed in Reference [13]. The encoding is performed in three steps, which we describe below. The first step is to create two sets of HVs, *ID* HVs and level HVs. Both ID HVs and *level* HVs are D-dimensional HVs where each element is either −1 or 1. The encoding scheme assigns a unique channel *ID* HV to each feature position. *ID*s are hypervectors that are randomly generated such that all features will have orthogonal channel *ID*s, i.e., $\delta(ID_i,\ ID_j) < 5{,}000$) for $D = 10{,}000$ and $i \neq j$; where the δ measures the element-wise similarity between the vectors. The HD computing encoder also generates a set of *level* HVs to consider the impact of each feature value. To create these level hypervectors, we compute the minimum and maximum feature values among all data points, $\mathbf{v}_{min}$ and $\mathbf{v}_{max}$, then quantize the range of $[\mathbf{v}_{min}, \mathbf{v}_{max}]$ into m levels. Each level is then assigned a corresponding level HV: $\mathbf{LV} = \{\mathbf{LV}_1, \ldots, \mathbf{LV}_m\}$. To encode a feature vector, the encoder looks at each position of the feature vector and element-wise multiplies the channel ID (ID_i) with the corresponding level hypervector (hv_i). The following equation shows how an n-length feature vector is mapped into the HD space with this encoding scheme:

$$\begin{gathered} H = [hv_1 * ID_1 + hv_2 * ID_2 + \cdots + hv_n * ID_n], \\ hv_j \in \{LV_1, LV_2, \ldots, LV_m\}, 1 \leqslant j \leqslant m, \\ ID_i \in \{-1, 1\}^D, LV_j \in \{-1, 1\}^D. \end{gathered}$$

(2) Training: The simplicity of HD training makes it distinguished from conventional learning algorithms. Consider hypervector $\mathcal{H}_i$ as the encoded hypervector of input i with the procedure explained above, which required the inner-product of $\mathcal{D}$ bit hypervectors followed by dimension-wise addition of n 1 bit values, where n is the number of features. Each input i belongs to a class j, so we further annotate $\mathcal{H}_i^j$ to show the class j of input i, as well. HD training simply adds all hypervectors of the same class to generate the final model hypervector. Therefore, the class hypervector of label j, denoted by C^j, is

$$C^j = \mathcal{H}_0^j + \mathcal{H}_1^j + \cdots = \sum_k \mathcal{H}^j. \tag{1}$$

Meaning that we simply accumulate the encoded hypervectors for which their original input belongs to class j.

Another advantage of HD over DNNs is HD supports efficient one-pass training, i.e., visiting each input just once and adding the $\mathcal{H}_i$s to create the model yields acceptable accuracy, while DNN training requires hundreds of iterations over the whole data set to converge to the final accuracy. HD accuracy can also be improved by **retraining** the model. During retraining, the encoded hypervector of each input is created again, and its similarity with the existing class (model) hypervectors is checked (see step 3). If a *misprediction* is observed, say that encoded $\mathcal{H}^j$ belonging to class C^j is predicted as class C^k, then the model is updated as follows, which means the information of $\mathcal{H}^j$ causing (mis)-similarity to C^k is discarded:

$$\begin{aligned} C^j &= C^j + \mathcal{H}^j, \\ C^k &= C^k - \mathcal{H}^j. \end{aligned} \tag{2}$$

(3) Similarity checking: The inference step as well as the retraining step need to find out the most similar class hypervector to the encoded one. Most commonly, this is performed by cosine similarity while other metrics (e.g., Hamming distance) could be appropriate depending on the problem:

$$cos(\vec{\mathcal{H}}, \vec{C}^j) = \frac{\vec{\mathcal{H}} \cdot \vec{C}^j}{\|\vec{\mathcal{H}}\| \cdot \|\vec{C}^j\|}. \tag{3}$$

Equation (3) shows the similarity checking of encoded hypervector $\mathcal{H}$ with class hypervector C^j. Since classes are constant, $\|\vec{C}^j\|$ can be pre-calculated. $\|\vec{\mathcal{H}}\|$ can be factored out as it is common for all candidate classes to be compared with $\mathcal{H}$. Hence, cosine similarity reduces to a simple dot-product between $\mathcal{H}$ and C^js. These vectors are *not* in binary, they are the results of accumulating several other binary vectors.

2.2 Hyperdimensional Computing Clustering

The HD Clustering algorithm is very similar to the popular K-means algorithm [16]. The first step of HD Clustering, like Classification is to first encode the data into high-dimensional space. In this article, we evaluate two encodings, both covered in Section 2.1. HD Clustering then operates on the encoded HVs as the main datatype. HD Clustering, like K-means, then selects random centers to start. HD Clustering then iterates through all of the encoded data points while comparing them with the cluster centers using a similarity metric and assigning each point to the center it is most similar to. In K-means, that similarity metric is the Euclidean distance. In HD, we utilize cosine similarity for non-binary values, but Euclidean distance could also be used. However, HD maps data into high-dimensional space, $D = 10{,}000$, so calculating cosine similarity is much more efficient. After all the points are labeled, the new centers are chosen and the process is repeated until convergence or the maximum number of iterations is reached. Convergence occurs when no point is assigned to a different cluster compared to the previous iteration. The main difference is that HD Clustering adds a pre-processing step to the Clustering algorithm that maps the data into high-dimensional space, or hypervectors.

2.3 Related Work

HD computing is light-weight enough to run with acceptable performance on CPUs [17]. However, utilizing a parallel architecture can significantly speed up HD execution time. Imani et al. showed two orders of magnitude speed up when HD runs on GPU [6]. Salamat et al. proposed a framework that facilitates fast implementation of HD algorithms on FPGA [7]. Due to the bit-level operations in HD, which is more suitable for FPGAs than GPUs, they claimed up to 12× energy and 1.7× speed up over GPUs. HD requires much less memory than DNNs, but the required memory capacity is still beyond the local cache of many devices. Thus, an excessive amount of energy and time is spent moving data between these devices and their main memory (off-chip memory in the case of FPGAs).

To resolve this, prior work used PIM architectures, where processing occurs in memory, eliminating the time and energy of data movement [18–20]. In FELIX [8], a *digital* PIM architecture was proposed. However, digital PIM operations are significantly slower than equivalent analog PIM operations. Prior work accelerated the inference phase of HD computing in analog PIM with an associative memory [6]. However, the associative memory only stored the trained class hypervectors, so the input data needed to be encoded elsewhere and then moved into the associative memory, negating the benefit of less data movement. Also, the associative memory only supports inference in HD. In this article, we implement HD Computing in an analog PIM ReRAM architecture based on ISAAC [11]. This architecture allows us to fully implement HD Computing operations end-to-

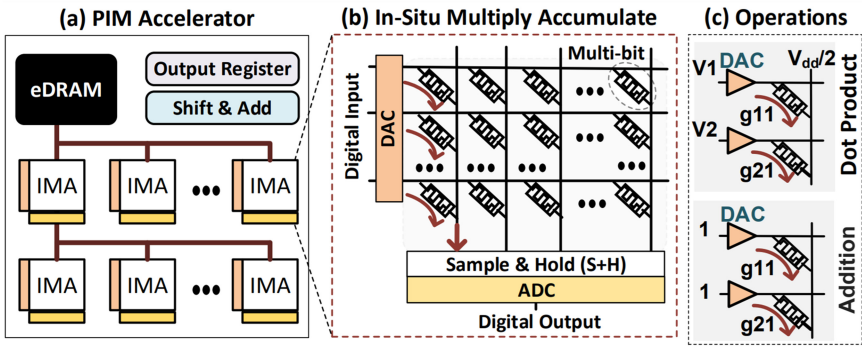


Fig. 2. Overview of the PIM architecture used by HyDREA.

end from encoding to inference unlike prior work. Our architecture differs in that we further take advantage of HD Computing's robustness to noise and relax the precision of the ADCs. We target the ADCs as they are the highest energy overhead in the architecture [11, 21].

Several works claimed that HD signal representations are inherently robust to various forms of noise [22–25]. Work in Reference [23] investigated the robustness of HD to RTL level errors (e.g., bit-flips) during computation and found an HD-based approach tolerating an 8.8× higher probability of bit-level errors. Similar results are reported in Reference [26].

Work in Reference [23] presented preliminary evidence showing that HD delivered superior performance to conventional data representations in the presence of bit-level errors during processing. Similarly, bit-level errors occur during data transmission as a result of channel noise and interference from multiple users. To the best of our knowledge, there has been no systematic empirical (or theoretical) evaluation of HD as an avenue for achieving robust learning when data must be communicated over noisy channels. This article compares HD computing with a "Federated learning" approach for training other ML models and proposes a new analog PIM architecture to accelerate the whole HD computing algorithm from training to inference.

3 HyDREA ANALOG PIM ARCHITECTURE

Combining the energy savings by eliminating data movement and a parallel architecture suitable for dimension-wise parallelism of HD algorithms, analog PIM, with its simple arithmetic support, appears as a promising solution for HD computing. A PIM architecture needs to support three classes of in-memory operations; (1) dot-product for the matrix multiplication in encoding and the similarity metric in inference, i.e., the $\vec{\mathcal{H}} \cdot \vec{C}^j$ part in Equation (3), in which each dimension of $\mathcal{H}$ and C^j is fixed-point (results of binary vector additions), (2) addition and subtraction for training and retraining where, as explained by Equation (1), we add $\mathcal{H}_i^j$s to produce C^j, which denotes the final class hypervector of inputs with label j, and (3) search operation to find the best matched class in inference, by finding the maximum of cosine similarity scores between the encoded query $\mathcal{H}$ and all class hypervectors. The baseline architecture provided by ISAAC [11] is perfect for mapping HD Computing to an analog PIM architecture, because it supports all three of the above operations. This can be seen in Figure 2(c).

(1) Dot Product: The top half shows how the dot product operation is implemented in our analog PIM crossbar. Assume each resistive cell in the first (i.e., the shown one) column is programmed to resistances R_{11} and R_{21} where R_{ij} belongs to row i and column j. Voltages V_1 and V_2 are applied to the first and second rows. The corresponding generated current flows through the column is I_1, which shows the result of dot-product. A larger I shows larger number, and since $I = {}^{V}/_{R}$, the

resistance of memristive cells need to be proportional to the *inverse* of the value they represent. For 2D vectors, A and B, the first set of inputs, A, is programmed into the resistances R_{11} and R_{21} having the conductances of ($A_{11} = 1/R_{11} = C_{11}$ and $A_{21} = 1/R_{21} = C_{21}$). Afterwards, the second set of inputs, B, is applied as the voltages at each row ($B_{11} = V_1$ and $B_{21} = V_{21}$). As the figure shows, by applying input values as the voltages to the rows and storing values as conductances, Ohm's law dictates that the current flowing through each resistor is the product of the conductance and applied voltage. Following Kirchhoff's law, the current accumulated at each column is equal to the sum of all the currents flowing through resistors of the column. That is, the total current is $I_1 = C_{11} \cdot V_1 + C_{21} \cdot V_2$. For our design, we store the class hypervectors as the conductances of the ReRam matrix and the query HV is sent as the DAC input voltages.

(2) Addition: The bottom half of Figure 2(c) shows how the addition is implemented in a crossbar analog PIM architecture. Addition works analogous to the dot product, except all the input voltages are set to logical 1 (i.e., V_{high}). This, the aggregate current of passing through the first column is $I_1 = C_{11} + C_{21}$.

(3) Search: Upon performing dot-product between the query hypervector with all class hypervectors, the search operation needs to find the class with the maximum similarity score. In analog PIM, search is implemented using nearest distance search, which finds the most similar value for a given reference. However, we desire a search for the maximum value (so the reference is unknown). But we know the maximum value of the cosine similarity metric is 1, hence we can implement our maximum value search with the already supported nearest distance search by searching for the value that has the highest similarity to reference 1. Hence, the returned value will be maximum score. Note that similarity check returns the closest value (absolute difference) by prioritizing MSB bits.

HyDREA takes advantage of HD computing's robustness to noise to reduce computational complexity without losing a significant amount of accuracy. By reducing the bitwidth of the ADCs in analog PIM, HyDREA is able to achieve significant energy savings. However, it comes at a cost of inaccurate computations. However, HD computing is robust to hardware failures and inaccurate computations, making it a perfect candidate to be accelerated by our design. With our bitwidth reduction optimization, HyDREA is able to achieve the energy efficiency of digital PIM with the speed of analog PIM.

3.1 Architecture

Figure 2(a) shows the architecture HyDREA constituting of multiple **In Situ Multiply Accumulate (IMA)** blocks. In our implementation, HyDREA comprises 24 IMA blocks. The design choice of using 24 IMA blocks was to ensure that our architecture can fit the largest dataset tested. This is critical, because if all the data does not fit, data would need to be offloaded and stored off chip. The load and store operations in our ReRAM array are very costly and would incur a significant amount of latency to our design. IMA blocks are memory crossbars with the capability of performing analog addition and dot-product operations. Each IMA block consists of 8 crossbar arrays, each of which contains 128 rows and 128 columns of memory cells. There are 8×128 **Digital-to-Analog (DAC)** blocks per IMA, i.e., 128 per each crossbar arrays, allocated to the rows to convert the incoming digital signal (voltage) to analog (current) to perform computation. There is also a shared **Sample and Hold (S+H)** block, and shared Analog-to-Digital (ADC) blocks in each IMA. Figure 2(b) shows an example of a crossbar memory array. Each bitline is connected to all the wordlines through memresistive cells, which have stored the information (e.g., values of class dimensions) by changing the resistance level of each cell. Each memresistive cell in our configuration is a 2 bit MLC, i.e., it has four resistance states to be able to represent 2 bits. Storing the HD model, i.e., the values of classes dimensions, needs to program the NVMs, which is a slow write operation.

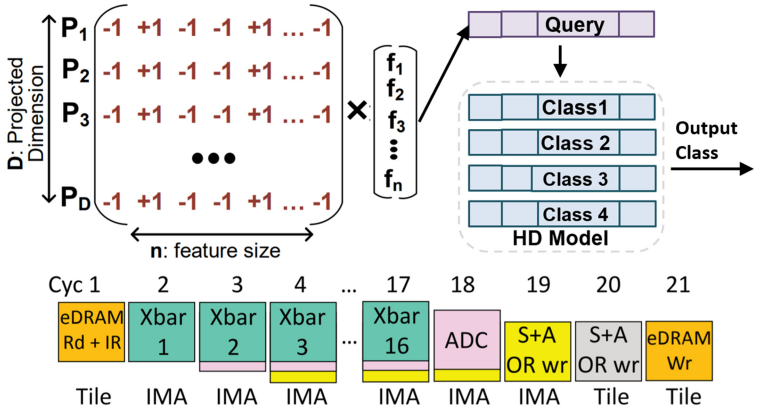


Fig. 3. Example of inference in HyDREA.

However, it is only done one time before beginning the inference step, so the overhead is amortized in the entire course of inference.

Figure 3 shows an example of how inference is performed in HyDREA. The first step is to encode the input. The input is stored in the eDRAM buffer of the encoder tile. When a new input shows up, it allows the current input to proceed with its next operation. This operation is itself pipelined (shown in Figure 3). In the first cycle, an eDRAM read is performed to read the input. These values are sent over the shared bus to the IMA for the encoder and recorded in the **input register (IR)**. After the input values have been moved, the IMA will perform the matrix multiplication during the next 16 cycles.

In the next 16 cycles, the eDRAM is ready to receive other inputs and deal with other IMAs. Over the next 16 cycles, the IR feeds 1 bit at a time for each of the input values to the crossbar arrays. The first 128 bits are sent to crossbars 0 and 1, and the next 128 bits are sent to crossbars 2 and 3. At the end of each cycle, the outputs are latched in the Sample and Hold circuits. In the next cycle, these outputs are fed to the ADC units. The results of the ADCs are then fed to the shift-and-add units, where the results are merged with the **output register (OR)** in the IMA.

As shown in Figure 3, at the end of cycle 19, the OR in the IMA has its final output value. This is sent over the shared bus to the central units in the tile. The central OR contains the final results for encoding at the end of cycle 20. During this time, the IMA for the next input has already begun processing to maintain utilization. Finally, in cycle 21, contents of the central OR are written to the eDRAM that will provide the inputs for the similarity check. The similarity check is then performed with the same pipeline as it too is a matrix multiplication.

3.2 Challenges

To perform the computation in analog, PIM needs to convert the signals into analog domain. For this, it requires to employ DAC and ADC converters at the inputs and outputs, respectively. As shown in previous work, these signal domain converters contribute to a significant overhead in the residing architecture [11, 21], which reaches up to 89% of the system power consumption. However, the overhead of these converters can be significantly alleviated as it is exponentially tied in the precision of converters. This, obviously, increases the error as the signal levels are quantized. Fortunately, it is less problematic in the context of HD computing thanks to its remarkable tolerance to error, as information is spread over all the independent and identically distributed dimensions of vectors, so failing the computation on a certain portion of dimensions (bits) should not affect the overall result noticeably.

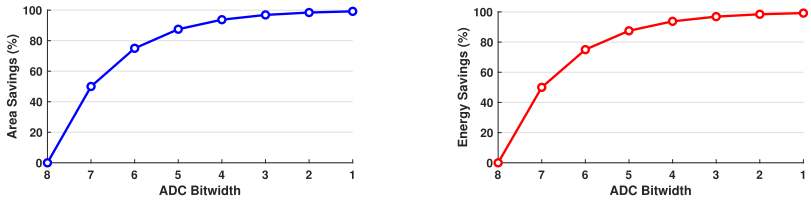


Fig. 4. Area savings (a) and energy consumption savings (b) as the bitwidth of the ADC is dropped.

Furthermore, the addition of ADCs for conversion is the largest overhead of using analog PIM for computation. The ADCs take up a huge amount of area as with each bit of resolution added, their area doubles. Prior work tried to alleviate this by sharing the large ADC across multiple blocks [11]. This approach can slow down computation. However, in this article, we significantly reduce this overhead by using extremely low precision ADCs (as low as 2-bits), which our application, HD Computing, can handle.

3.3 HyDREA: **Analog PIM Architecture Optimiztions**

ADC Reduction: As in Section 3.2, the energy overhead of conversion from the digital domain to the analog domain and back dominates the energy usage of analog PIM, and this is handled by the ADC blocks. Thus, our task to improve the energy efficiency of analog PIM focuses on improving the energy efficiency of the ADC blocks. We achieve this by reducing the precision of the ADC blocks. Figure 4 shows the expected energy and area savings of reducing the bitwidth of an ADC. The results from the energy breakdown of ISAAC shows 89% of energy is used on ADC conversion. Then, knowing that each bit we drop from the ADC reduces ADC energy by approximately half, we can extrapolate the expected savings. As the figure shows, for each reduction in the bitwidth of an ADC, we expect the area and energy consumption to halve. This is because to add support for each additional bit, the amount of circuit area doubles and therefore, the energy usage approximately doubles. Instead of using 8-bit ADC blocks in analog PIM that achieve full precision conversion to the digital domain, if we reduce the bitwidth of the ADCs, then we can reduce the energy usage by half for every bit of the ADC we drop. This will save a significant amount of energy during the analog to digital conversion step in analog PIM. However, as mentioned, our computations will lose accuracy, and as we drop more bits, our computations will become more inaccurate as we sacrifice precision for energy efficiency.

We can reduce our ADC blocks from 8 bits to n bits. By doing this, we will convert the first n most significant bits and omit the $8 - n$ least significant bits. For example, if we use a 6 bit ADC block to convert 167, then we would lose the last two bits and output 164 instead. This leads to good approximate conversions with large numbers but very poor approximation with smaller numbers. If we use a 6 bit ADC block to convert 7, then we would get 4, which is almost 50% off. Furthermore, we do not produce inaccurate conversions every time. If we convert 172 with a 6 bit ADC block, then we wold get 172, because the last two bits of 172 are both 0. Therefore, we produce exact computations when the bits we would drop are all zero. Our ADC block conversions fall into three categories: exact conversions, slightly inaccurate conversions, and highly inaccurate conversions. Since HD computing utilizes dot product as the similarity check, the larger computations dominate the dot product operation and therefore, the highly inaccurate conversions of smaller operations do not effect the accuracy of the HD model. Therefore, we are able to take advantage of reducing the bitwidth of ADCs to create an analog PIM architecture for accelerating HD computing that does not incur a significant loss in accuracy.

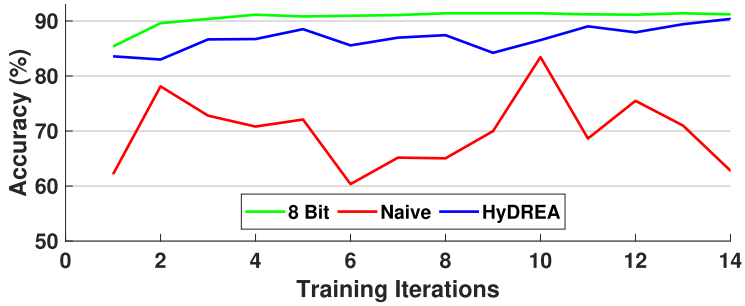


Fig. 5. Impact of HyDREA using a 4 bit model on training compared to training a naive bitwidth reduction 4 bit model and training a 8 bit model.

DAC Reduction: We additionally reduce the energy and execution time overhead of analog PIM by reducing the number of DACs and IMA blocks needed. We achieve this by reducing the precision of the HD model bitwidth.

Due to HD computing's robustness to noise, we could simply reduce the bitwidth of the HD model and achieve efficiency gains without a significant drop in accuracy. When reducing the bitwidth further, training the HD model becomes unstable and the accuracy does not converge. Figure 5 compares training an HD model with 4 bits of precision and training the same model with a full 8 bits of precision. The details of the setup and software used to obtain these results can be found in Section 5. The top line shows that training an 8 bit model is much smoother and clearly improves in each iteration compared to training with reduced bitwidth. This is because, as HVs are added up and adjusted with retraining, some dimensions may saturate the available bitwidth. Any additional change to dimensions with saturated bitwidths that attempt to change the dimension in the direction of the bitwidth saturation does not improve the model further. For instance, when using a bitwidth of 4, the maximum positive value a dimension can represent is 7. If during retraining the dimension would be increased further, then it would instead stay at 7. In contrast, if the dimension is adjusted with subtraction, then it would decrease normally despite any previous attempts to increase the dimension further. This causes over-adjustments in the HD model during retraining when an abnormal change is applied. This is why the accuracy does not converge during retraining with greatly reduced bitwidths. HyDREA is able to improve upon the naive design of simply reducing the bitwidths by additionally modifying the HD algorithm to complement the bitwidth reduction.

As explained in Section 2, the HD model is initially trained by adding up all of encoded data points into one class HV for each class. When reducing the bitwidth of the HD model from 8 bits to 4 bits, 4 bits may not provide enough precision for model convergence during retraining, preventing the HD model from performing effectively at lower bitwidths. To subvert this problem, we propose to analyze the initial HD model to identify key dimensions that need to utilize the full bitwidth available. HyDREA then locks these dimensions to either the maximum or minimum value to ensure the the HD model does not drastically change during retraining.

We propose that the largest dimensions in both the positive and negative directions that saturate the desired bitwidth are key dimensions, as dot product is used as the similarity metric. Hence, the largest dimensions in both positive or negative direction contribute the most to the resulting dot product. Dimensions with the largest values in either direction show that most data points from that class agree in that dimension, i.e., a class HV that represents the class well should ensure these dimensions are not over-adjusted.

To support bitwidth reduction, we propose to modify the initial training algorithm of HD. To identify key dimensions in the HD model to lock, our design first performs the initial training with a full 8 bit representation. HyDREA copies the initial class HV and takes the absolute value of all the dimensions in the class HV and finds the indices of the largest α dimensions that would saturate the desired bitwidth. They are set to the maximum (minimum) value if they saturated in the positive (negative) direction. The other dimensions are scaled down to the desired bitwidth. This is done for all k class HVs. The initial model is then loaded into our PIM architecture. The dimensions that were previously set to the maximum or minimum value are locked from changes during retraining to prevent the HD model from over adjustments. HyDREA only locks dimensions that would saturate the desired bitwidth. If the dimensions do not saturate the desired bitwidth, then the bitwidth is sufficient and no change is needed. This lock is achieved by not enabling the write bits at locked dimensions.

Figure 5 compares training an HD model with the naive approach of simply reducing the bitwidth to 4 and training the same model with HyDREA using the same bitwidth. The graph shows how HyDREA improves upon the naive design, as during retraining the model is clearly improving and increasing in accuracy like the full 8 bit model. Meanwhile, the naive design's accuracy fluctuates greatly and does not converge.

3.4 HyDREA: **Supporting HD Clustering**

Here, we discuss how we can use HyDREA to support HD Clustering. As described in Section 2, the HD Clustering algorithm is very similar to the K-means algorithm with a different similarity metric. For HD Clustering, instead of using Euclidean distance, we use cosine similarity to measure the distance between the samples and the cluster centers. This makes mapping HD Clustering onto our existing architecture relatively simple as for Classification, HyDREA already accelerates the similarity checking part of HD inference. Additionally, we use the same encodings for Clustering and Classification, so that accelerator can be reused as well. Therefore, to map HD Clustering to HyDREA, we feed the samples in the original feature domain into our encoding block. Then, to update the distances between the samples and the cluster centers, we feed the cluster centers into the inference accelerator as the class HVs and the samples as the query HVs. This then gives us both the distance in cosine similarity between each sample and all the cluster centers as well as the cluster that each sample is most similar to. The next step of the HD Clustering algorithm, which is to chose the next cluster centers is too complex to accelerate in PIM. However, 98% of the time is spent on encoding and similarity checking. Therefore, offloading updating the cluster centers to the host CPU does not incur a significant amount of overhead.

4 NETWORK SIMULATION

Figure 6 shows an overview of our federated learning framework and how devices communicate. There are two kinds of devices in our network edge devices and the central node. Edge devices are where local samples are generated. During training, they use a cut down version of our accelerator for HyDREA that just implements encoding to map the data into HD space. The sample is then sent to the central node, where on its way there, the encoded sample is subject to wireless communication noise. The central node's purpose is to collect all encoded samples from all of the edge nodes, train a global model, and perform inference. It too uses our accelerator, except it has full training and inference functionality. Once the global model is sufficiently trained, it can be used for inference. Upon inference, the edge device again encodes the input sample to HD space. The sample is then sent to the central node wirelessly incurring a varying degree of noise. The central node then performs inference on the trained HD model and sends the resulting label back to the edge device.

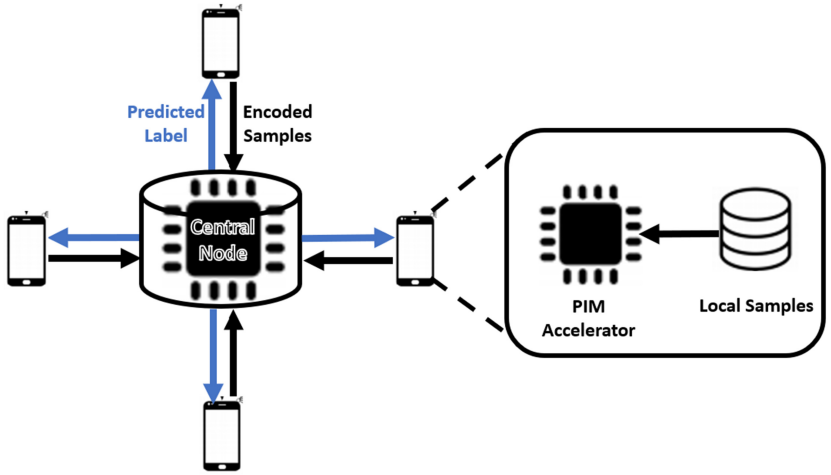


Fig. 6. An Overview of our framework for communicating in the federated learning environment.

We evaluate the feasibility of HyDREA in a "Federated learning" environment, by utilizing a popular network simulator—ns-3 [12]—to model the communication between devices and simulate wireless noise. In the Results section, we compare HyDREA with other ML algorithms in the same noisy environment. The ns-3 physical layer model calculates **bit error rates (BER)** taking into account the **Forward Error Correction (FEC)** present in WiFi standards such as IEEE 802.11a/g/n. The model first calculates the received **signal-to-noise ratio (SNR)** based on parameters used in the simulation model and then calculates a **packet error rate (PER)** based on the mode of operation (e.g., modulation, coding rate) to determine the probability of successfully receiving a frame (**packet success rate (PSR)**). The received signal SNR depends on the following parameters:

- Transmission powers of devices: Since noise power is usually constant, increasing the transmission power results in a higher SNR, thus lower BER. However, since energy efficiency is crucial in many applications, IoT devices usually operate in low power modes, resulting in low SNR.
- Distance between communicating nodes: As two communicating nodes get further away, the received signal strength decreases, resulting in low SNR.
- Propagation loss: The loss in the communication channel is different for different topologies. For example, if two devices are in the line-of-sight of each other, this scenario would incur much less loss compared to them communicating in a dense downtown with buildings blocking the view.
- Interference: When many devices communicate at the same time, each other's signals act as an interfering signal, which degrades the demodulation and decoding performance at the receiving end. In this case, we have to calculate **signal-to-interference-plus-noise ratio (SINR)**.

We study how HD Classification and Clustering performance changes with varying transmission power levels, distance, different propagation loss scenarios, and under different number of interfering devices. Additionally, the error rate depends on the modulation, coding and error correction mechanism adopted by the WiFi technology. Ns-3 allows us to study the error rates for modulation schemes such as BPSK, QPSK, 16–1024 QAM, under binary convolutional coding for rates $\frac{1}{2}$, $\frac{2}{3}$, $\frac{3}{4}$, and $\frac{5}{6}$. We can both enable or disable **forward error correction (FEC)** in all of these

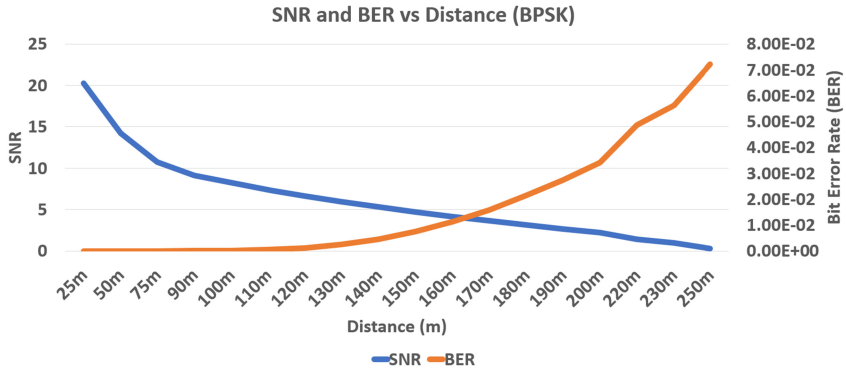


Fig. 7. SNR/BER vs. distance for BPSK modulation with Friis prop. loss.

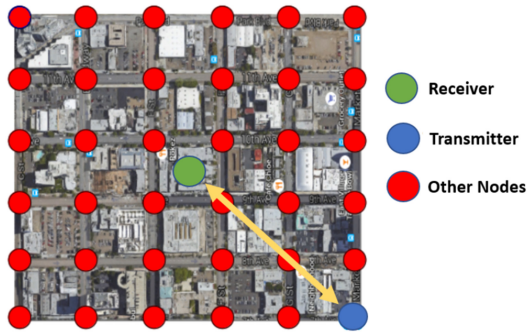


Fig. 8. Model of a downtown topology represented in ns-3, where buildings have higher signal attenuation compared to open-air and they block the line-of-sight when they are placed between the transmitters (blue) and the receiver (green).

cases. Our experiments use the WiFi protocol stack (802.11n), which is the most matured communication standard implementation in ns-3. There are efforts on modeling low-rate and low-power standards for IoT, but they are not fully developed yet. Hence, we modify the 802.11 PHY and MAC layer parameters and scale data rate and power values to imitate communication in an IoT environment. The modulation techniques and coding schemes of 802.11n, namely, BPSK, M-ary QAM, and **Direct-Sequence Spread Spectrum (DSSS)**, are common with many low-power wireless protocols [27]. Different techniques have different SNR versus BER (Bit error rates) curves, but these curves are the same across protocols [28–30]. Since we adjust the parameters of 802.11n, we can simulate the characteristics of low power IoT protocols by operating at the low SNR regions of the SNR-BER curve. We vary the distance between the transmitter and the receiver to collect data at various SNRs. We evaluate with the Friis propagation loss model. Figure 7 shows the BER versus distance curve between transmitter and receiver. We additionally test error rates from other sources of noise. Such as a downtown scenario with buildings in between the nodes shown in Figure 8 or a highly congested network. We use the hybrid building propagation loss model consisting of Okumura-Hata [31], ITU-R 1411 and ITU-R 1238 [32] loss models. The model includes the multi-path fading loss through building walls for both **line-of-sight (LoS)** and no LoS cases. There are also random communication attempts between other nodes in the network resulting in dynamic BER and packet losses.

Table 1. Dataset Information

Dataset	Type	# Classes	# Train Data	# Test Data	# Features
UCIHAR [34]	**Classification**	6	6,213	1,554	561
CARDIO [35]	**Classification**	2	1,913	213	21
FACE [36]	**Classification**	2	22,441	2,494	608
ISOLET [37]	**Classification and Clustering**	26	6,238	1,559	617
Hepta [38]	**Clustering**	7	N/A	212	3
Tetra [38]	**Clustering**	4	N/A	400	3
Two Diamonds [38]	**Clustering**	2	N/A	800	2
Wingnut [38]	**Clustering**	2	N/A	1,016	2
Iris [38]	**Clustering**	3	N/A	135	3

We compare HD with two baseline approaches. In the first, we assume that corrupted data packets are discarded and must be re-transmitted. This ensures the accuracy of the resulting model, but increases latency and energy consumption—especially in congested networks. Second, we train on the corrupted data. This eliminates the need to re-transmit packets but may slow model convergence or cause the model to converge to a worse local optimum (recall that Neural Networks are a non-convex optimization problem). Due to the robustness of HD Computing to noise, the HD model is able to learn more effectively from corrupted packets than other ML models, eliminating the need to re-transmit data while ensuring a high-quality result. Low-power networks such as LoRaWAN and LPWAN usually operate at very low SNRs [33], which can result in error rates ranging from 10^{-5} to 10^{-1}. Many applications require perfect data reconstruction at the receiver, so it is often aimed for networks by design to have an error rate at upper levels of this range. We show that HD is very resilient to errors, such that one can deliberately use very low-power for communication and operate at extremely low SNRs, going beyond the error rates that of standard network configurations, while still getting acceptable accuracy for the learning tasks. This comes with large energy savings that is crucial for resource-constrained IoT devices. We additionally compare HD Computing robustness to **Error Correction Codes (ECC)** in wireless communication in Section 5.10.

5 EVALUATION

5.1 Experimental Setup

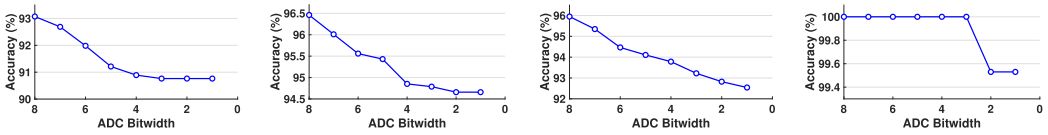
We verified the functionality of HyDREA using both software and hardware implementations. In software, we implemented HD Classification and Clustering on an Intel Core i7 7600 CPU using an optimized C++ implementation. For the hardware implementation, we used an analog-based PIM architecture proposed in Reference [11]. We modify the ISAAC architecture to more efficiently run for HD Computing by relaxing the bitwidth resolution of the ADCs. Our PIM design works at 1.2 GHz and uses n bit ADCs, 1 bit DACs, and 128×128 arrays, where each memresistor cell stores 2 bits. To estimate the energy consumption and execution time of HyDREA, we utilize the detailed energy and execution time breakdown of an ISAAC tile found in the original ISAAC paper [11]. We then calculate the estimated execution time and energy by summing up the required operations for HD Computing. We tested our approach for HD Classification on four practical Classification applications and for HD Clustering on six datasets from the Fundamental Clustering Problem Suite [38], shown in Table 1.

5.2 HyDREA and Dimensionality

To test the impact of dimensionality on HD Classification and Clustering robustness, we utilized the 6.64 SNR test with all datasets. Table 2 summarizes the results, where each entry in the table is

Table 2. Impact of Dimensionality and Data Representation on the Robustness of HD Computing Classification and Clustering Accuracy

Dimensionality	10,000	8,000	6,000	4,000	2,000
RP Binary (Classification)	0.58%	0.82%	1.44%	1.89%	2.39%
ID-Level Binary (Classification)	0.56%	0.79%	1.52%	1.78%	2.42%
RP (Clustering)	0.58%	2.31%	2.65%	2.86%	3.24%
ID-Level Binary (Clustering)	0.66%	2.48%	2.52%	2.79%	3.13%
ID-Level Int (Clustering)	44.89%	46.60%	64.71%	72.82%	72.13%
ID-Level Float (Clustering)	85.17%	85.19%	85.23%	85.43%	85.55%



(a) ISOLET (b) UCIHAR (c) FACE (d) CARDIO

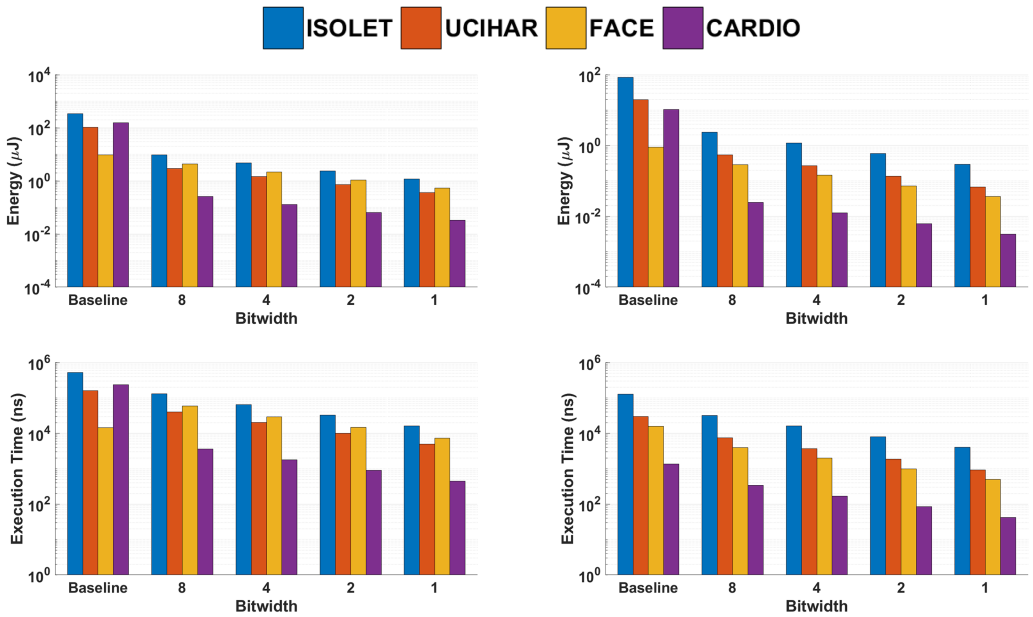
Fig. 9. Impact of bitwidth reduction on accuracy of HyDREA.

the average accuracy for all datasets at that dimensionality. There is a clear relationship between HD robustness to errors and dimensionality. One may think that we can achieve faster execution and lower energy consumption with lower dimensionality; but due to our PIM's highly parallel nature, as long as the HD model fits into the PIM arrays, execution time and energy does not change. Since our design requires a highly robust HD model, the rest of our tests utilize a dimensionality of $D = 10{,}000$. Additionally, the table shows that the data representation highly impacts the robustness of HD. Binary values are the most robust, because each individual bit flip impact the correctness of the end result the same. However, with other representations such as floating point, depending on the bit flipped, the error can increase significantly. For instance, if an exponent bit is flipped, that would incur significantly more error than if a mantissa bit was flipped. For the most robust models, one should transmit binary encoded HVs.

5.3 HyDREA and the Impact of our Analog PIM Architecture on HD Classification

Figure 9 shows the impact of ADC bitwidth reduction on HD model accuracy for four practical applications. The accuracy of each model reduces as the bitwidth drops, but not significantly. When the ADC bitwidth is 4, the average accuracy drop across all applications is 1.5%. This is because our ADC blocks provide highly accurate approximations for high value conversions, and the high value numbers dominate the dot product output. Thus, the resulting dot product closely approximates the exact version. Also, the resulting dot product does not need to be exact, owing to HD's robustness to hardware inaccuracies. Despite inaccurate results, the classes are separated enough that slight variations still result in the HD model selecting the same output class. Overall, HyDREA reduces bitwidth to 2 while only losing 1.8% in accuracy.

Figure 10 shows the impact of our analog PIM architecture with 2 bit ADCs and varying model bitwidths on energy consumption and execution time. Our proposed architectural changes drastically improve the energy efficiency and execution time of HD. Our proposed architecture uses 2 bit ADCs and 1 bit models, and achieves 32× (29×) speed up and 232× (267×) higher energy efficiency than the baseline architecture during inference (retraining). Also, in high SNR cases, these models achieve comparable accuracy to full precision models.



(a) Retraining (b) Inference

Fig. 10. Energy consumption and execution time of HyDREA using different model bitwidths during training and inference with an ADC bitwidth of 2.

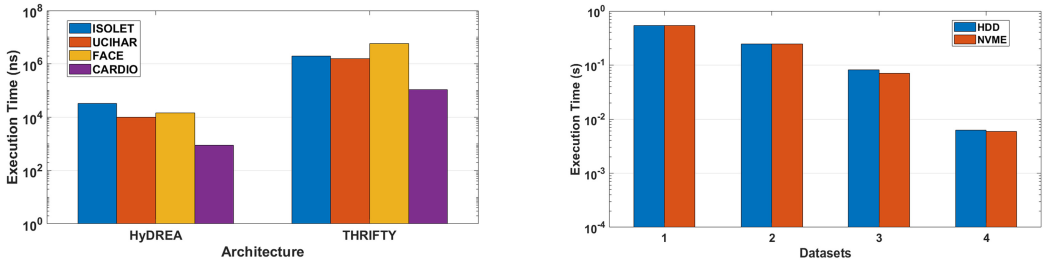
Table 3. Speedup of HyDREA Over a Digital PIM Implementation with the Same Bitwidth as HyDREA with the Same Area

Dataset	ISOLET	UCIHAR	CARDIO	FACE
Retraining	110.4×	111.8×	105.6×	115.2×
Inference Same Bit Digital	128.9×	137.3×	139.9×	136.1×

5.4 HyDREA Versus Processing in Storage and Digital Processing in Memory

Figure 11 compares HyDREA execution time during the training process to THRIFTY [39]. The results show that due to the slower digital operations in THRIFTY, as well as the higher latency of computing near flash storage, HyDREA is on average 180× faster during training than in storage computing. Furthermore, Figure 11 also compares the impact of high bandwidth memory, or specifically NVME storage, on HD Computing latency. We perform this test on the same machine where the only difference is for HDD, we store all data on a slow spinning hard drive and for NVME, we use a PCIe generation 4.0 NVME storage drive. The results clearly show that the higher bandwidth does not impact the overall latency of HD Computing. Therefore, in storage computing solutions such as THRIFTY do not have much to gain from utilizing NVME technologies. Thus, analog processing in memory architectures such as HyDREA are more capable of delivering faster execution times than digital processing in memory architectures.

In Table 3, we also compare HyDREA with a FELIX [8] digital PIM-based implementation of HD Computing. We compare using the same model bitwidths and memory area. Our results show that HyDREA is 111× faster than the digital PIM design during retraining and 136× faster than the digital PIM design during inference on average, because the individual operations in analog PIM are much faster than they are in digital PIM. HyDREA achieves better speed up during inference



(a) Comparison With THRIFTY [39]

(b) Impact of NVME on HD

Fig. 11. Execution time comparison of HyDREA with THRIFTY, a processing in storage architecture for HD Computing and the impact of higher bandwidth memories such as NVME on HD Computing.

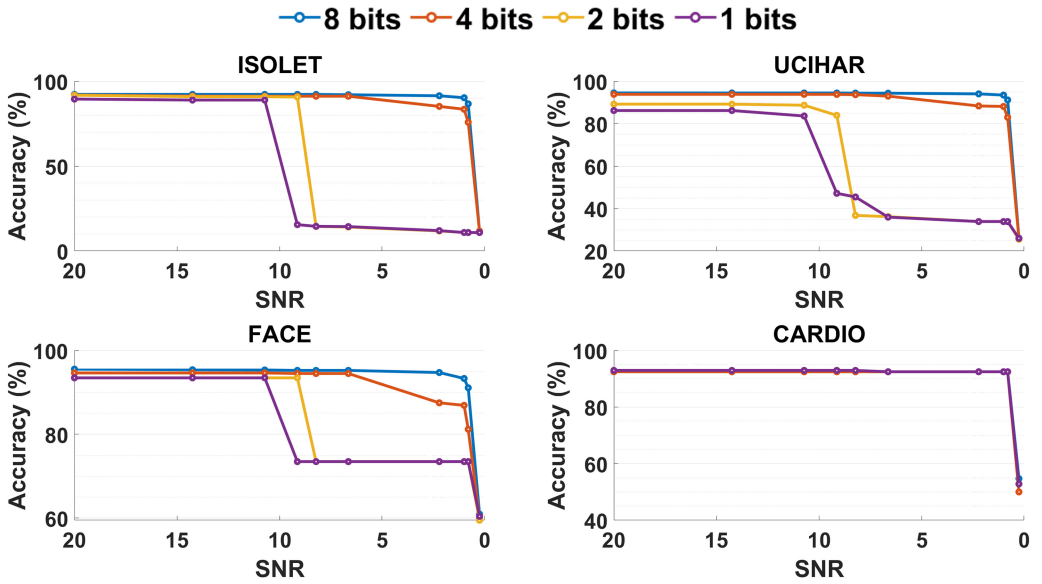


Fig. 12. Accuracy of Design as the SNR varies with an ADC bitwidth of 2 and varying model bitwidth.

than retraining when compared to digital PIM, because inference only involves the dot product operation while retraining includes addition operations to adjust the HD model. Due to relying on nor-based operations in digital PIM, execution time scales quadratically for multiplications. Therefore, because analog PIM directly implements multiply and accumulate, HyDREA achieves better speed up during inference and retraining.

5.5 HyDREA and the Impact of SNR on HD Classification

Figure 12 shows the impact of SNR on model accuracy in our analog PIM architecture. We can load in low bitwidth models when the channel has a high SNR to achieve the best energy consumption and execution time. However, during high network traffic, longer communication distance, or other factors that incur a high amount of noise on the wireless channel, we need to load in the higher bitwidth models to maintain accuracy. This is because our highly quantized models are taking advantage of HD's robustness to noise by effectively adding more noise to the computation. Therefore, if the environment, in this case wireless communication, is also adding noise, the robust

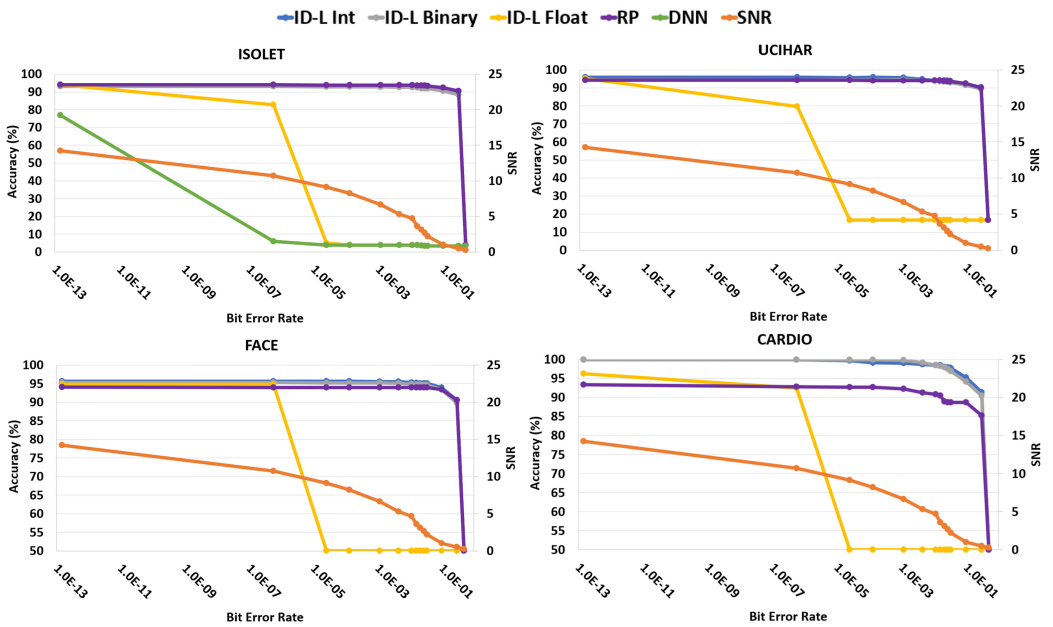


Fig. 13. Accuracy of HD Classification as the SNR varies with different encodings and data representations.

property of HD does not hold up. However, if we adaptively switch which model is loaded based on the SNR, then we can maintain high accuracy and achieve significant energy and execution time savings when possible.

Figure 13 shows the impact of SNR on model accuracy for two different encodings as well as different datatype representations. Results from all datasets show a similar pattern with increasing bit error rate. HD using integer and binary hypervectors is much more robust to noise as compared to floating-point representations. Since floating-point numbers are represented with mantissa and exponent, if the exponent bits are flipped because of an error, then the number itself changes significantly. We additionally compare to a DNN for the ISOLET dataset [40]. The DNN model uses a 16bit floating point representation for its weights, so we can observe the same problem with robustness in DNNs. The data also demonstrates that the **random projection (RP)** encoding offers similar robustness to noise as the ID-level encoding with binary values. This is likely because our implementation of random projection also encodes hypervectors to binary values (through a final sign function), so both the random projection and quantized ID-level encodings lead to similarly robust binary hypervectors. Last, random projection achieves on average, the same accuracy as ID-level, but beats ID-level in some datasets, such as ISOLET, while loses in accuracy to others, such as CARDIO and EMG, as both of them are time-series signals, which random projection does not classify well.

5.6 HD Versus Other Classifiers

We also compared HD to state-of-the-art classifiers (**Linear Regression (LR)**, **MultiLayer Perceptron (MLP)**, Perceptron, **Support Vector Classification (SVC)**) and evaluated its robustness to noise on our four datasets. Figure 14 shows the results for (1) data with no noise, and (2) data corrupted with SNR of 2.21. We choose an SNR or 2.21, because it is the worst practical scenario in our ns-3 setup. All classifiers have comparable accuracy with no noise. While HD stays robust with a significant amount of noise, the other classifiers become very inaccurate. The high-dimensional

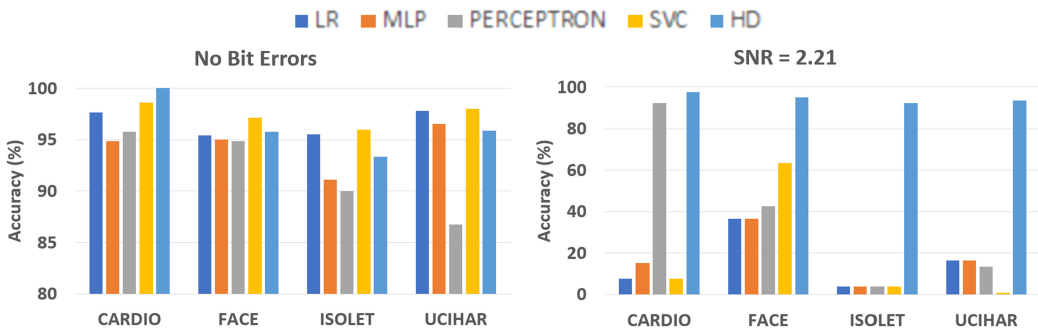


Fig. 14. Comparison of the robustness of HD to other classifiers.

Table 4. Comparison of HyDREA with the State-of-the-Art DNN PIM Accelerator Q-PIM [41]

Design	Exact Accuracy	2.21 SNR Accuracy	Latency(s)	Energy(J)
HyDREA	93.4%	92.1%	$9.98 \times 10^{-6} s$	$8.02 \times 10^{-7} J$
Q-PIM [41]	98.5%	10%	$4.1 \times 10^{-3} s$	$4 \times 10^{-4} J$

nature of the hypervectors used in HD leads to significant redundancy in representation, which improves its robustness to noise by 48× compared to other classifiers at 2.21 SNR. In other words, HD loses 48× less accuracy compared to the other classifiers. This gives us a metric where noise robustness is defined by how well the model maintains accuracy with the added wireless noise.

5.7 HyDREA Versus State-of-the-Art PIM DNN Accelerator

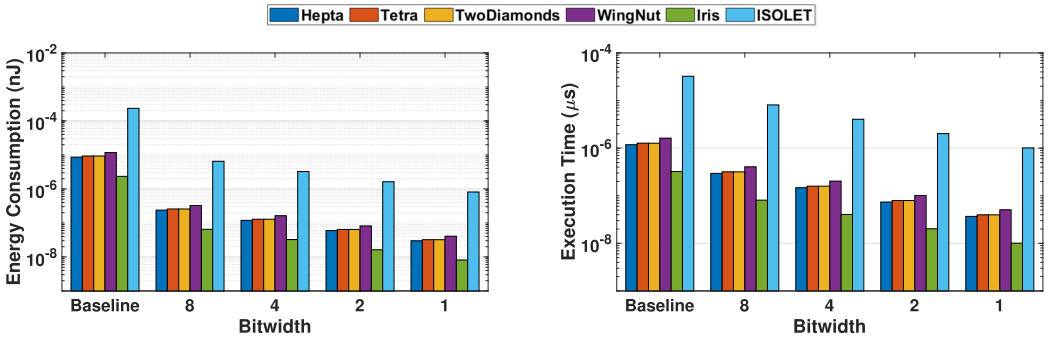
In Table 4, we compare HyDREA with a State-of-the-Art DNN PIM accelerator Q-PIM [41]. The results show that State-of-the-Art DNNs are able to achieve higher accuracy on more complex datasets such as MNIST. However, in the presence of wireless communication errors, HD Computing is able to maintain its accuracy, while traditional DNNs become unreliable and return random classification results. Furthermore, due to HD Computing's light weight operation, HyDREA achieves a 411× speedup and 498× energy efficiency improvement over Q-PIM.

5.8 HyDREA Architecture Impact on Clustering Energy Consumption and Execution Time

Figure 15 shows the impact of our analog PIM architecture with 2 bit ADCs and varying model bitwidths on energy consumption and execution time for HD Clustering. Our proposed architectural changes drastically improve the energy efficiency and execution time of HD Clustering. Our proposed architecture uses 2 bit ADCs and 1 bit models, and achieves 32× speed up and 289× higher energy efficiency than the baseline architecture during Clustering. Also, in high SNR cases, just like for Classification, these models achieve comparable accuracy to full precision models.

5.9 HD Clustering Accuracy and Robustness Versus K-means

We also compared HD to a state of the art Clustering algorithm, K-means, and evaluated its robustness to noise. As can be seen from Figure 16, K-means has a comparable accuracy to HD when there are no bit errors in the dataset. To measure Clustering accuracy, we use a metric based on the mutual information between the cluster assignments returned by our algorithm and ground truth cluster labels. The metric is one when the predicted labels are perfectly correlated with the ground truth and zero when they are totally uncorrelated. Although accuracy is similar without



(a) Energy (b) Execution Time

Fig. 15. Energy consumption and execution time of HyDREA for one Clustering iteration using different model bitwidths with an ADC bitwidth of 2.

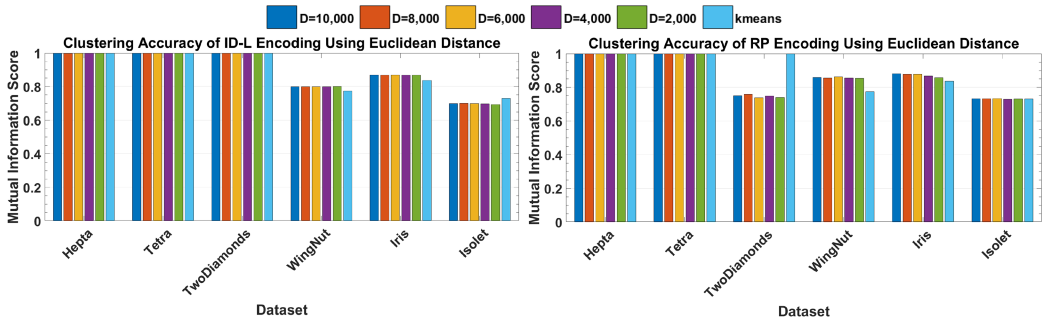


Fig. 16. Comparison of HD clustering with K-means accuracy with no bit errors.

errors, when we introduce errors HD Clustering is significantly more robust. Our proposed system also looses less than 1% in the mutual information score, even in scenarios with an SNR under 7 dB, which is 57× more robust to noise than K-means.

Figure 17 compares HD Clustering vs K-means Robustness to bit error rates. K-means has similar robustness to bit error rates as HD using integer and floating point representations, until a breaking point around 10^{-3} bit error rate for most datasets. This is especially clear with the Isolet dataset, which is the biggest dataset we use. HD Clustering is able to maintain accuracy for much larger bit error rates than K-means when running Isolet. HD gains this additional robust property from the high-dimensional nature of the hypervectors used in HD computing leading to significant redundancy in the representation, which improves robustness to noise similar to our Classification results.

Additionally, similar to our Classification results, the results from all datasets show a similar pattern where HD using integer and binary hypervectors is much more robust to noise as compared to floating-point. Since floating-point numbers are represented with mantissa bits and exponent bits, if the exponent bits are flipped because of an error, the number itself changes significantly, thus incurring more noise. Integer representation performs closer to binary. Random projection provides similar accuracy to binarized ID-Level as random projection encodes hypervectors to binary values as well. Binary representation is the most robust as each individual bit flip incurs the same proportion of noise.

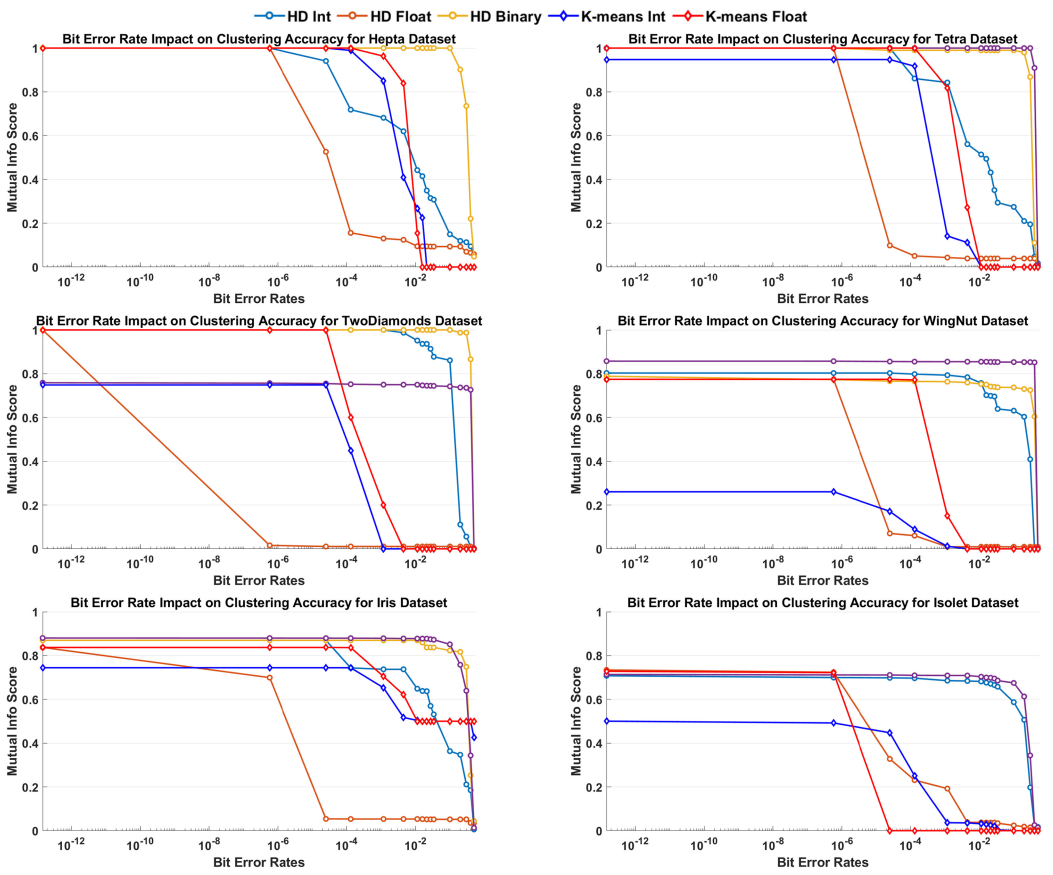


Fig. 17. Accuracy of HD clustering as the SNR varies with different encodings and data representations vs. K-means.

5.10 Impact of Bit Error Rates on Decoding

Some HD Computing encoding methods have the property where the encoded HV can be decoded back into the original feature vector. For instance, with access to the ID and LV HV banks used to encode the HV in ID-Level, one can decode the encoded HV to get back the original feature vector with some errors [42]. In Figure 18, we show the impact of dimensionality on the quality of the recovered feature vector using the ID-Level encoding. The y-axis shows the mean-squared-error of the original feature vector with the decoded one. We test against a range of bit error rates that could be seen in wireless communication as well as across different dimensions. The results indicate that with higher dimensionality, we are able to recover a better quality sample in the original feature space. Additionally, as the bit error rate increases, our decoding quality decreases. The decoded feature vectors become drastically different after bit error rates of around 0.001 for both 5,000 and 10,000 dimensions.

5.11 HD Computing Versus Error Correcting Codes (ECC)

In conventional systems, the transmitter performs three steps to generate the wireless signal from data: source coding, channel coding, and modulation. First, a source encoder removes the redundancies and compresses the data. Then, to protect the compressed bitstream against the

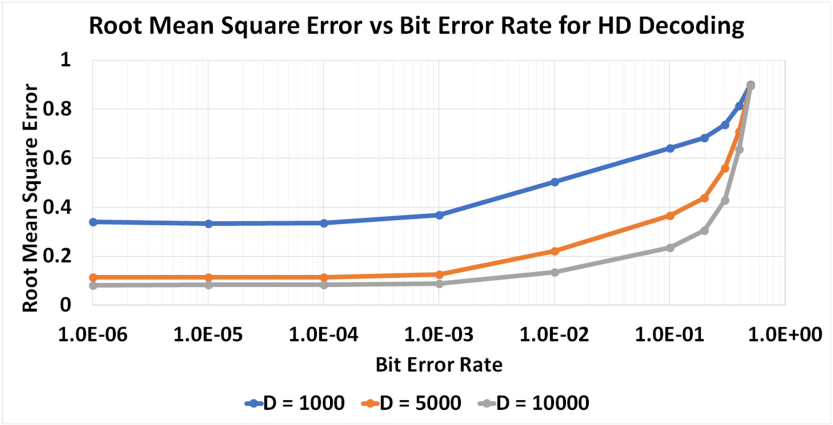


Fig. 18. Impact of dimensionality on decoding quality.

impairments introduced by the channel, a channel code is applied. The coded bitstream is finally modulated with a modulation scheme that maps the bits to complex-valued samples (symbols), transmitted over the communication link. The receiver inverts the above operations, but in the reverse order. A demodulator first maps the received complex-valued channel output to a sequence of bits. This bitstream is then decoded with a channel decoder to obtain the original compressed data; however, it might be possibly corrupted due to the channel impairments. Last, the source decoder provides a (usually inexact) reconstruction of the transmitted data by applying a decompression algorithm.

In this work, we deal with robust learning over unreliable communication channels, so we focus only on the channel coding techniques from this pipeline for our comparison. **Error correcting codes (ECC)** are used in channel coding for controlling errors in data over unreliable and noisy communication channels. The central idea is the sender encodes the message with redundant information in the form of an ECC. This redundancy allows the receiver to detect a limited number of errors that may occur anywhere in the message, and often to correct these errors without retransmission. We implement the setups depicted in Figure 19 and compare channel codes to our method. We refer to the framework shown in Figure 6 with the evaluation setup described in Section 4, and evaluate the inference robustness of the different communcation systems. For all experiments, we have an **Additive White Gaussian Noise (AWGN)** channel, over a range of SNR values, and the modulation type is QAM. In the first setup, there is no channel coding and raw data samples are transmitted over the channel. The HD classifier at the receiver side uses these raw data samples corrupted by bit errors to do inference. In the second setup, we add channel coding to the configuration. In the third setup, we apply HD encoding to data at the transmitter side and transmit hypervectors. In this case, we do not need to do encoding at the receiver, only a simple similarity check for HD Inference on the corrupted hypervectors suffices. In the fourth setup, we add channel coding on top of HD encoded hypervectors to further add redundancy.

In Figure 20(a), we compare a rate $\frac{1}{2}$ convolutional channel code with HD encoding. Viterbi decoder is used to decode the transmitted bitstreams at the centralized receiver. Both channel codes and HD encoding are applied directly to raw data samples, as illustrated in second and third communication setups, respectively. The results show that HD encoding has better performance at similar coding rates than convolutional codes. At 35% BER, HD still has around 90% accuracy with 10k dimension hypervectors whereas convolutional code quickly loses accuracy then completely fails. In Figure 20(b), we compare HD encoding with high-dimension hypervectors to using

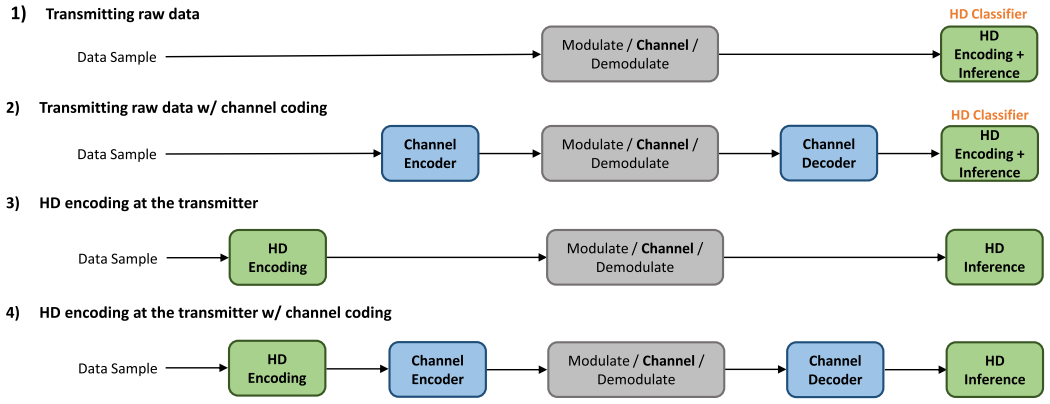


Fig. 19. Simulated communication setups.

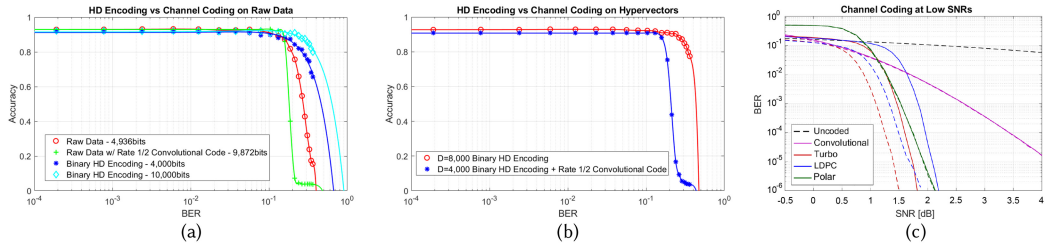


Fig. 20. (a) Comparison of HD encoding to channel coding (setup 1, 2, and 3), (b) combined HD encoding and channel coding (setup 3 and 4), (c) channel coding performance at low SNRs, exact (dashed) and approximate (solid) decoding algorithms.

channel codes combined with lower dimension hypervectors. HD encoding alone performs better at the same overall coding rate, meaning that channel codes do not provide extra protection to the hypervectors. The above results can be explained by Figure 20(c), for which we refer to Reference [43]. All the plotted coding methods are rate $\frac{1}{2}$ as the convolutional code used in the previous experiments. We show the SNR versus BER curves for both the exact (dashed) and approximate (solid) decoding algorithms of the considered methods. As implied by the plots, channel coding gains are significant at moderate to high SNRs. However, BER performance of channel coding converges to that of uncoded communication at low SNRs, for which we perform our experiments. In such cases, particularly where BER is greater than 10%, HD encoding is more robust. Moreover, channel codes aim at correcting the errors and reconstructing the original data. Since we are only interested in using the received data for classification or clustering, the exact reconstructions are not necessarily needed. HD encodings are more suitable for this purpose, as the holographic representation property allows to maintain as much information as possible when part of the data is lost.

6 CONCLUSION

In this article, we proposed HyDREA, an HD computing system that is Robust, Efficient, and Accurate. We proposed a PIM architecture that adaptively changes the bitwidth of the model based on the SNR of the incoming sample to maintain the robustness of the HD model while achieving high accuracy and energy efficiency. Our results indicate that our proposed system loses less than 1% Classification accuracy even in scenarios with an SNR under 7 dB. Our PIM architecture is

also able to achieve 255× better energy efficiency and speed up execution time by 28× compared to the baseline PIM architecture. We evaluated the feasibility of HyDREA in a "Federated learning" environment, by utilizing a popular network simulator, ns-3, to model the communication between devices and simulate wireless noise. We compared HyDREA with other light-weight ML algorithms in the same noisy environment. Our results demonstrated that HyDREA is 48× more robust to noise than other comparable ML algorithms. We additionally tested the robustness of HD Clustering in the same network simulation scenarios and found that our proposed system also looses less than 1% in the mutual information score, even in scenarios with an SNR under 7 dB, which is 57× more robust to noise than K-means. Finally, we extended our PIM architecture to support Clustering and our results show that we are able to achieves 289× higher energy efficiency and 32× speed up compared to the baseline architecture during Clustering.

REFERENCES

[1] Jakub Konečnỳ, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. Retrieved from https://arXiv:1610.05492.

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

[3] Mohsen Imani et al. 2021. Revisiting hyperdimensional learning for FPGA and low-power architectures. In *Proceedings of the IEEE Symposium on High-Performance Computer Architecture (HPCA'21)*. IEEE.

[4] Mohsen Imani, Chenyu Huang, Deqian Kong, and Tajana Rosing. 2018. Hierarchical hyperdimensional computing for energy efficient classification. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, 108.

[5] Mojan Javaheripi, Mohammad Samragh, Tara Javidi, and Farinaz Koushanfar. 2020. GeneCAI: Genetic evolution for acquiring compact AI. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'20)*. ACM, 350–358. DOI: http://dx.doi.org/10.1145/3377930.3390226

[6] Mohsen Imani, Abbas Rahimi, Deqian Kong, Tajana Rosing, and Jan M. Rabaey. 2017. Exploring hyperdimensional associative memory. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'17)*. IEEE, 445–456.

[7] Sahand Salamat et al. 2019. F5-HD: Fast flexible FPGA-based framework for refreshing hyperdimensional computing. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'19)*. ACM, 53–62.

[8] Saransh Gupta et al. 2018. FELIX: Fast and energy-efficient logic in memory. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'18)*. ACM, 55.

[9] Mohsen Imani et al. 2020. DUAL: Acceleration of clustering algorithms using digital-based processing in-memory. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. IEEE, 356–371.

[10] Mohammad Samragh, Mojan Javaheripi, and Farinaz Koushanfar. 2020. Encodeep: Realizing bit-flexible encoding for deep neural networks. *ACM Trans. Embed. Comput. Syst.* 19, 6 (2020), 1–29.

[11] Ali Shafiee et al. 2016. ISAAC: A convolutional neural network accelerator with in situ analog arithmetic in crossbars. In *Proceedings of the International Symposium on Computer Architecture (ISCA'16)*. IEEE, 14–26.

[12] Thomas R. Henderson, Mathieu Lacage, George F. Riley, Craig Dowell, and Joseph Kopena. 2008. Network simulations with the ns-3 simulator. *SIGCOMM Demonst.* 14, 14 (2008), 527.

[13] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. 2017. VoiceHD: Hyperdimensional computing for efficient speech recognition. In *Proceedings of the International Conference on Rebooting Computing (ICRC'17)*. IEEE, 1–6.

[14] Mohsen Imani et al. 2019. BRIC: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'19)*. IEEE, 1–6.

[15] Abbas Rahimi, Simone Benatti, Pentti Kanerva, et al. 2016. Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition. In *Proceedings of the IEEE International Conference on Rebooting Computing (ICRC'16)*. IEEE, 1–8.

[16] Mohsen Imani et al. 2019. HDCluster: An accurate clustering using brain-inspired high-dimensional computing. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'19)*. IEEE/ACM.

[17] Mohsen Imani et al. 2019. A binary learning framework for hyperdimensional computing. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'19)*. IEEE/ACM.

[18] Chao Li et al. 2020. A scalable design of multi-bit ferroelectric content addressable memory for data-centric computing. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'20)*. IEEE.

[19] Mohsen Imani et al. 2019. Floatpim: In-memory acceleration of deep neural network training with high precision. In *Proceedings of the International Symposium on Computer Architecture (ISCA'19)*. IEEE, 802–815.

[20] Mohsen Imani et al. 2020. Deep learning acceleration with neuron-to-memory transformation. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'20)*. IEEE, 1–14.
[21] Soroush Ghodrati, Hardik Sharma, et al. 2020. Mixed-signal charge-domain acceleration of deep neural networks through interleaved bit-partitioned arithmetic. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. 399–411.
[22] Abbas Rahimi, Pentti Kanerva, Luca Benini, and Jan M. Rabaey. 2018. Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of ExG signals. *Proc. IEEE* 107, 1 (2018), 123–143.
[23] Abbas Rahimi, Pentti Kanerva, and Jan M. Rabaey. 2016. A robust and energy-efficient classifier using brain-inspired hyperdimensional computing. In *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, 64–69.
[24] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. 2017. Voicehd: Hyperdimensional computing for efficient speech recognition. In *Proceedings of the IEEE International Conference on Rebooting Computing (ICRC'17)*. IEEE, 1–8.
[25] Mohsen Imani et al. 2019. QuantHD: A quantization framework for hyperdimensional computing. IEEE Trans. Comput.-Aided Design Integr. *Circ. Syst*. 39, 10 (2019), 2268–2278.
[26] Haitong Li et al. 2016. Hyperdimensional computing with 3D VRRAM in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'16)*. IEEE, 16–1.
[27] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. 2017. Low power wide area networks: An overview. IEEE Commun. *Surveys Tutor*. 19, 2 (2017), 855–873.
[28] IEEE. 2009. 802.11n-2009—IEEE Standard for Information technology—Local and metropolitan area networks. Retrieved from https://standards.ieee.org/standard/802_11n-2009.html.
[29] IEEE. 2020. 802.15.4-2020—IEEE Standard for Low-Rate Wireless Networks. Retrieved from https://standards.ieee.org/standard/802_15_4-2020.html.
[30] Theodore S. Rappaport et al. 1996. Wireless Communications: Principles and Practice. Vol. 2. Prentice Hall PTR, New Jersey.
[31] Arturas Medeisis and Algimantas Kajackas. 2000. On the use of the universal Okumura-Hata propagation prediction model in rural areas. In *Proceedings of the IEEE 51st Vehicular Technology Conference Proceedings (VTC'00)*, Vol. 3. IEEE, 1815–1818.
[32] International Telecommunication Union. [n.d.]. Retrieved from https://www.itu.int//.
[33] Orion Afisiadis, Matthieu Cotting, Andreas Burg, and Alexios Balatsoukas-Stimming. 2019. On the error rate of the LoRa modulation with interference. *IEEE Trans. Wireless Commun.* 19, 2 (2019), 1292–1304.
[34] UCI Machine Learning Repository. [n.d.]. Retrieved from https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities.
[35] UCI Machine Learning Repository. [n.d.]. Retrieved from https://archive.ics.uci.edu/ml/datasets/cardiotocography.
[36] Gregory Griffin, Alex Holub, and Pietro Perona. 2007. Caltech-256 object category dataset. California Institute of Technology.
[37] UCI Machine Learning Repository. [n.d.]. Retrieved from http://archive.ics.uci.edu/ml/datasets/ISOLET.
[38] Alfred Ultsch. 2005. U* C: Self-organized clustering with emergent feature maps. In *Proceedings of the Lernen, Wissensentdeckung und Adaptivit GI Workshops (LWA'05)*. Citeseer, 240–244.
[39] Saransh Gupta, Justin Morris, Mohsen Imani, Ranganathan Ramkumar, Jeffrey Yu, Aniket Tiwari, Baris Aksanli, and Tajana Šimunić Rosing. 2020. THRIFTY: Training with hyperdimensional computing across flash hierarchy.
[40] Mohammad Samragh, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2017. Customizing neural networks for efficient FPGA implementation. In *Proceedings of the IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. IEEE, 85–92.
[41] Yun Long, Edward Lee, Daehyun Kim, and Saibal Mukhopadhyay. 2020. Q-pim: A genetic algorithm based flexible dnn quantization method and application to processing-in-memory platform. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–6.
[42] Mohsen Imani, Yeseong Kim, Sadegh Riazi, John Messerly, Patric Liu, Farinaz Koushanfar, and Tajana Rosing. 2019. A framework for collaborative learning in secure high-dimensional space. In *Proceedings of the IEEE 12th International Conference on Cloud Computing (CLOUD'19)*. IEEE, 435–446.
[43] Bashar Tahir, Stefan Schwarz, and Markus Rupp. 2017. BER comparison between convolutional, turbo, LDPC, and polar codes. In *Proceedings of the 24th International Conference on Telecommunications (ICT'17)*. IEEE, 1–7.

Received 15 July 2021; revised 24 February 2022; accepted 3 March 2022