HyDREA: Towards More Robust and Efficient Machine Learning Systems with Hyperdimensional Computing

Justin Morris, Kazim Ergun, Behnam Khaleghi, Mohsen Imani, Baris Aksanli, Tajana Rosing

***Abstract*—Today's systems, especially in the age of federated learning, rely on sending all the data to the cloud, and then use 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 paper, we propose HyDREA, a HD computing system that is Robust, Efficient, and Accurate. To evaluate the feasibility of HyDREA in a federated learning environment with wireless communication noise, we utilize NS-3, a popular network simulator that models a real world environment with wireless communication noise. We found that HyDREA is $48\times$ more robust to noise than other comparable ML algorithms. We additionally propose a Processing-in-Memory (PIM) architecture that adaptively changes the bitwidth of the model based on the signal to noise ratio (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 of 6.64. Our PIM architecture is also able to achieve $255\times$ better energy efficiency and speed up execution time by $28\times$ compared to the baseline PIM architecture.**

I. 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 which 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", HD computing offers three benefits. 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. Second, local training of the HD model is extremely simple and more energy efficient than many existing ML techniques [2]. 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 GPUs [2], FPGAs [3], and PIM architectures [4]. Although GPUs and FPGAs provide a suitable degree of parallelism, the complexity of their resources, e.g., floating point units or DSP blocks, is by 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.

In this paper, we propose HyDREA, a HD computing system that is Robust, Efficient, and Accurate. We evaluate the feasibility of HyDREA in a "Federated learning" environment, by utilizing a popular network simulator – NS-3 [5] – 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 $48\times$ more robust to noise than other comparable ML algorithms. We additionally propose 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. Our PIM architecture is also able to achieve $255\times$ better energy efficiency and speed up execution time by $28\times$ compared to the baseline PIM architecture.

II. 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.

A. Hyperdimensional Computing

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.

(1) Encoding: Let us assume a feature vector $\mathbf{F} = \{f_1; f_2; \dots; f_n\}$, with n features ($f_i \in \mathbb{N}$) in original domain. The goal of encoding is to map this feature vector to a D dimensional space vector: $\mathbf{H} = \{h_1; h_2; \dots; h_D\}$. The encoding first generates D dense bipolar vectors with the same dimensionality as original domain, $\mathbf{P} = \{\mathbf{p}_1; \mathbf{p}_2; \dots; \mathbf{p}_D\}$, 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 $i-th$ dimension of the encoded data as:

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

where $sign$ is a sign function which 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:

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

(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 $\mathcal{C}^j$, is:

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

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 $\mathcal{C}^j$ is predicted as class $\mathcal{C}^k$, the model is updated as follows, which means the information of $\mathcal{H}^j$ causing (mis)-similarity to $\mathcal{C}^k$ is discarded.

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

(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.

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

Equation (3) shows the similarity checking of encoded hypervector $\mathcal{H}$ with class hypervector $\mathcal{C}^j$. Since classes are constant, $\|\mathcal{C}^j\|$ can be pre-calculated. $\|\mathcal{H}\|$ 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 $\mathcal{C}^j$s. These vectors are *not* in binary, they are the results of accumulating several other binary vectors.

B. Related Work

HD computing is light-weight enough to run with acceptable performance on CPUs [6]. 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 [2]. Salamat et al. proposed a framework that facilitates fast implementation of HD algorithms on FPGA [3]. 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. In FELIX [4], 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 [2]. 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.

Several works claimed that HD signal representations are inherently robust to various forms of noise [7], [8], [9], [10]. Work in [8] 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 [11].

Work in [8] 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 paper 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.

III. 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 $\mathcal{H} \cdot \mathcal{C}^j$ part in Equation 3 in which each dimension of $\mathcal{H}$ and $\mathcal{C}^j$ 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 $\mathcal{C}^j$ 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.

A. Architecture

Fig. 1(a) shows the architecture HyDREA constituting of multiple In-Situ Multiply Accumulate (IMA) blocks. In our implementation, HyDREA comprises of 24 IMA blocks so it can fit the largest benchmark. 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) in order to perform computation. There is also a shared Sample and Hold (S+H) block, and shared Analog-to-Digital (ADC) blocks in each IMA. Fig. 1(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. However, it is only done one time before beginning the inference step, so the overhead is amortized in the entire course of inference.

B. 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,

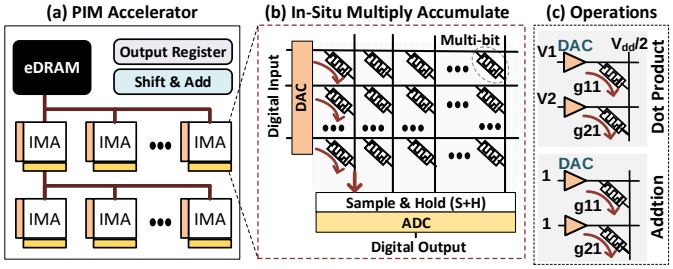


Fig. 1. Overview of the PIM architecture used by HyDREA

respectively. These signal domain converters contribute to a significant overhead in the residing architecture [12], [13], 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.

C. HyDREA: *Analog PIM Architecture Optimiztions*

ADC Reduction: As in Section III-B, 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. For each reduction in the ADC bitwidth, we expect the area and energy consumption to halve. This is because in order 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 (i.e. full precision), if we reduce the ADC bitwidth, 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, 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 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 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, 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

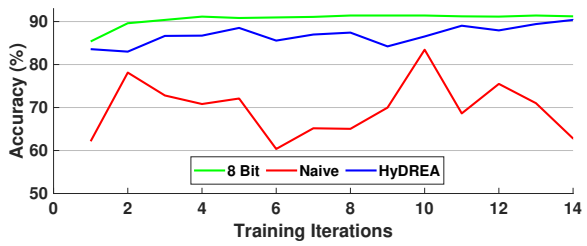


Fig. 2. Impact of HyDREA using a 4 bit model on training compared to training a naive bitwidth reduction 4 bit model and training a 8 bit model.

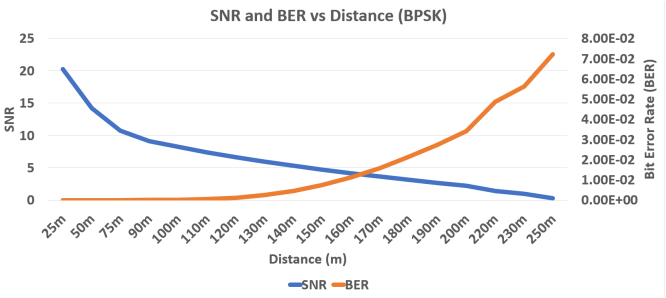


Fig. 3. SNR/BER vs distance for BPSK modulation with Friis prop. loss.

TABLE I
DATASET INFORMATION

Application	Dataset	# Classes	# Train Data	# Test Data	# Features
Speech Recognition [14]	**ISOLET**	26	6,238	1,559	617
Activity Recognition [15]	**UCIHAR**	6	6,213	1,554	561
Medical Diagnosis [16]	**CARDIO**	2	1,913	213	21
Face Detection [17]	**FACE**	2	22,441	2,494	608

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.

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.

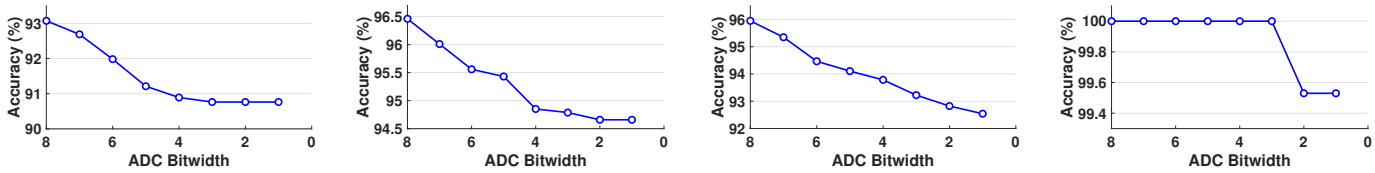
Due to HD computing's robustness to noise, we could simply reduce the bitwidth of the HD model and achieve efficiency gains without a significant drop in accuracy. When reducing the bitwidth further, training the HD model becomes unstable and the accuracy does not converge. Figure 2 compares training an HD model with 4 bits of precision and training the same model with a full 8 bits of precision. 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, it would instead stay at 7. In contrast, if the dimension is adjusted with subtraction, 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 II, 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 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, the bitwidth is sufficient and no change is needed. This lock is achieved by not enabling the write bits at locked dimensions.

Figure 2 compares training an HD model with the naive approach of simply reducing the bitwidth to 4 and training



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

Fig. 4. Impact of bitwidth reduction on accuracy of HyDREA.

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.

IV. Evaluation

A. Experimental Setup

We verified the functionality of HyDREA using both software and hardware implementations. In software, we implemented HD training and inference 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 [12]. Our PIM design works at 1.2GHz and uses n bit ADCs, 1 bit DACs, and 128×128 arrays, where each memresistor cell stores 2 bits. We tested the efficiency of our approach on four practical applications, shown in Table I.

We additionally study how HD performance changes with varying transmission power levels, distance, different propagation loss scenarios, and under a different number of interfering devices. To do this, we utilize the widely known network simulator, NS-3 [5]. The error rate depends on the modulation, coding, and error correction mechanism adopted by the wireless 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 ½, , ¾, . We test both with or without forward error correction (FEC).

These experiments use the WiFi standard (802.11n). The modes (High Throughput Modulation and Coding Schemes - HTMCSs) of 802.11n have different SNR vs BER (Bit error rates) curves. 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 3 shows the BER versus distance curve between transmitter and receiver.

B. HyDREA and Dimensionality

To test the impact of dimensionality on HD robustness, we utilized the 6.64 SNR test with all datasets. Table II summarizes the results. 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 $D = 10{,}000$.

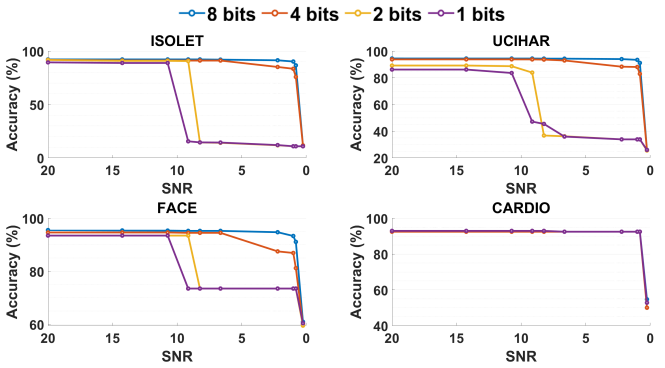
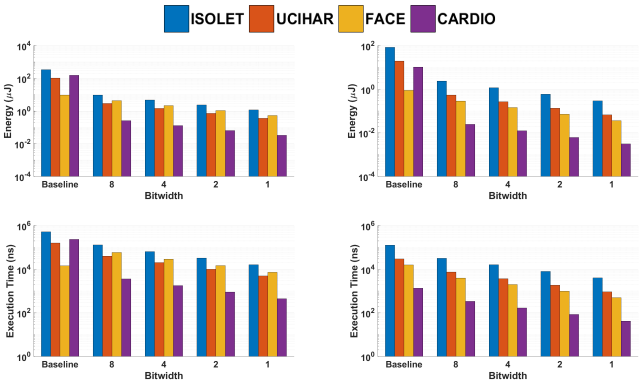


Fig. 5. Accuracy of Design as the SNR varies with an ADC bitwidth of 2 and varying model bitwidth.



(a) Retraining (b) Inference

Fig. 6. Energy consumption and execution time of HyDREA using different model bitwidths during training and inference with an ADC bitwidth of 2.

C. HyDREA and the Impact of our Analog PIM Architecture

Figure 4 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

TABLE II
Impact of Dimensionality on the Robustness of HD Computing

Dimensionality	10,000	8,000	6,000	4,000	2,000
Accuracy Loss	0.58%	0.82%	1.44%	1.89%	2.39%