CompHD: Efficient Hyperdimensional Computing Using Model Compression

Justin Morris, Mohsen Imani, Samuel Bosch, Anthony Thomas, Helen Shu, Tajana Rosing

***Abstract*—Hyperdimensional (HD) computing is a mathematical framework, inspired by neuroscience, which can be used to represent many machine learning (ML) problems. Data is first encoded into high dimensional space (on the order of 10^3 or 10^4 dimensions) to create hypervectors. HD computing combines these hypervectors to create a model used for inference. However, due to the high dimensionality of the hypervectors, inference in HD is very expensive, especially when it runs on embedded devices with limited resources. One naive approach to improve the efficiency of HD computing is to simply lower the dimensionality of hypervectors, which comes with a corresponding loss in accuracy. However, if the data is compressed intelligently, we can reduce the dimensionality of an HD model without sacrificing accuracy. To that end, we propose CompHD, a novel approach for compressing HD models while maintaining the accuracy of the original model. CompHD utilizes the mathematics of high-dimensional spaces to compress hypervectors into shorter vectors while maintaining the information of full length hypervectors. We evaluated the efficiency of CompHD on a variety of applications. Our results show that CompHD can reduce model size by an average of 69.7%, resulting in a execution time speed up of 4.1× and improving energy efficiency by 74% while maintaining the accuracy of the original model. This enables more low powered IoT devices to utilize HD computing for ML problems.**

I. Introduction

The emergence of the Internet of Things (IoT) has created an abundance of small embedded devices [1]. Many of these devices are used for cognitive tasks such as: face detection, speech recognition, image classification, activity recognition, etc. These devices want to run learning algorithms such as: deep Neural Networks (DNNs), (AlexNet [2], and GoogleNet [3] that provide excellent accuracy for cognitive tasks. However, these embedded devices have limited resources, such as limited battery power or limited memory [4]. Therefore, these devices are unable to run these resource intensive algorithms. To get around the resource limitation on embedded devices, many of them send the data they collect to a cloud server, which performs the resource intensive cognitive tasks. However, this is not desirable for many users due to security and network communication costs [5]. Thus, we need more efficient light-weight classifiers to perform cognitive tasks on embedded systems.

Brain-inspired Hyperdimensional (HD) computing has been proposed as a light-weight classifier to perform cognitive tasks on resource limited systems. Inspired by research from neuroscience, HD computing represents data as points in a high dimensional space. Past research utilized high dimension vectors (D $\geq$ 1,000), called hypervectors (HV), to represent neural activity in the brain. Prior work found that HD is able to provide high accuracy results for many cognitive tasks at a much lower computational cost than other learning algorithms [6], [7], [8], [9]. Work in [10] proposed a general encoding module that maps feature vectors into high-dimensional space while keeping most of the original data. Prior work also tried to design hardware acceleration for HD computing by mapping its operations into hardware, e.g., FPGA [11], [12], [13], [14], and tried to accelerate HD computing in hardware by binarizing the class hypervectors [15] or removing dimensions of the class hypervectors [16]. However, removing dimensions and binarizing the HD model causes accuracy loss because information captured by the HD model is removed. To achieve the best accuracy, HD computing needs to be trained with an integer model, but when using (D = 1,000) HVs to create and train the HD model, HD computing can still be too resource intensive for embedded systems.

Model size in HD computing is important because it increases the information storage capabilities of the model, resulting in better distinction between classes. However, with larger model sizes, there is a corresponding increase in computational costs. For example, when an HD model has D dimensionality and k classes, every query request costs $k * D$ additions and multiplications. To achieve acceptable accuracy, HD models typically use very high dimensionality. This is costly for embedded devices with limited resources. Inference requests take too long to calculate and additionally, the HD model may not fit into the main memory of embedded devices.

In this paper, we propose a robust and efficient solution to the computational complexity and spacial constraints of HD computing while maintaining comparable accuracy. Prior work reduced computation by simply lowering the dimensionality [7]. However, this approach leads to data loss in HD computing as the information in the dropped dimensions is no longer kept in the model. Our proposed HD computing framework, called CompHD, utilizes the mathematics of high dimensional spaces to reduce the size of the HD model and thus reduces the number of computations at inference time. CompHD splits up each class HV into s separate components and combines them into a reduced $d << D$ dimensional model. This method reduces the model size and number of computations by a factor of s while maintaining comparable accuracy to the original HD model. Using an empirical evaluation on several real world datasets, we show that CompHD can reduce the model size by an average of 69.7%, improve efficiency by 74%, and speed up execution time by $4.1\times$, while maintaining the accuracy of the original model. Our results show that CompHD enables more low powered IoT devices to solve ML problems with HD computing.

TABLE I
EFFECT OF REDUCING DIMENSIONALITY ON ACCURACY AND EXECUTION TIME

	Accuracy			Testing Time		
Dimension (D)	*1000*	*250*	*100*	*1000*	*250*	*100*
Activity Recognition	100%	5.3%	5.2%	49.4μ s	17.78μ s	9.88μ s
Valve Monitoring	100%	83.7%	51.0%	10.4μ s	3.74μ s	2.08μ s
Gesture Recognition	91.1%	84.4%	62.9%	13.0μ s	4.68μ s	2.6μ s

II. HIGH-DIMENSIONAL COMPUTING

Hyperdimensional computing is a computing paradigm involving long vectors with dimensionality in the thousands, called hypervectors [17]. There are several nearly orthogonal HVs in high-dimensional space [18]. HD combines these HVs with well-defined vector operations, while preserving most of the information from each individual HV. No one dimension in a HV has more responsibility to store any piece of information than any other component because HVs are holographic and (pseudo) random with i.i.d. components and a full holistic representation. The mathematics of high-dimensional space enable HD to be easily applied to different learning problems.

Figure 1 shows an overview of the structure of an HD model. HD computing consists of an encoder and a classifier. The encoder maps input data into hypervectors. The HVs are then combined to create one class HV to represent each class and stored in the classifier. The classifier uses the cosine similarity of any input HV with all of the class HVs to determine the output class. The class with the highest cosine similarity is selected as the output class.

A. Encoding

CompHD is a general framework that can be used to compress an HD model for any classification task. In IoT systems, devices usually get data from sensor nodes, which produce time-series data. Here we explain CompHD functionality in the context of time-series classification. We use an encoder designed for time series signals [19] to encode feature vectors into high-dimensional space. Our encoding first quantizes the feature values into m levels and assign a "level hypervector" $\mathcal{L}$ to each. The following equations show how the $\mathcal{L}s$ are used to encode a n length time series signal to generate the j^{th} training data HV in the i^{th} class where $\mathcal{N}$ is the length of the N-gram window, $\mathcal{G}_k$ is an intermediate HV that is calculated for each N-gram step and ρ^x is defined as a rotational shift to the right by x:

$$\mathcal{G}_k = [\rho^0(\mathcal{L}_1) + \rho^1(\mathcal{L}_2) + \ldots + \rho^{N-1}(\mathcal{L}_N)]$$

$$\mathcal{H}_i^j = [\mathcal{G}_1 + \mathcal{G}_2 + \ldots + \mathcal{G}_{n-N}]$$

B. Training

HD computing supports efficient one-pass training. To build a one-pass model, the encoder maps all training data to training HVs ($\mathcal{H}$). For all training HVs within a class

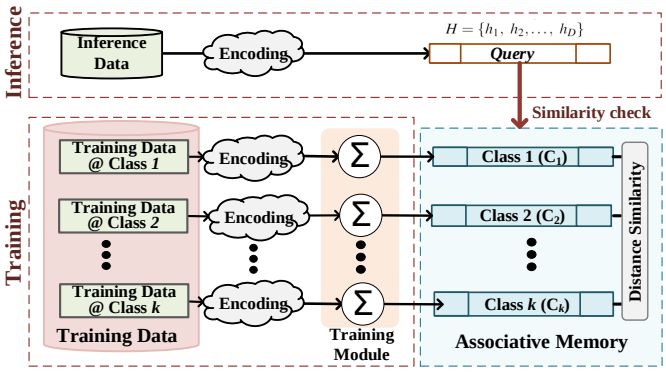


Fig. 1. Overview of creating an HD model and performing inference with an HD model

($\{\mathcal{H}_i^1, \mathcal{H}_i^2, \ldots, \mathcal{H}_i^j\}$), HD computing adds them together to create a single class HV ($\mathcal{C}_i$).

$$\mathcal{C}_i = \mathcal{H}_i^1 + \mathcal{H}_i^2 + \ldots + \mathcal{H}_i^j$$

Once this is done for every class, we have an HD model that can be used for inference. By creating a model in one pass through the training dataset, HD computing uses significantly less energy than other learning algorithms that need to take multiple passes over the training dataset to train a model. After training, all class HVs are stored in the classifier.

C. Associative Search

Upon inference, the encoder first maps the input data into a query HV ($\mathcal{Q}$), using the same encoding that was used to train the HD model. A similarity metric is used to determine the strength of a match between the query HV and each class HV. The most common metric used in HD computing is cosine similarity, but note that other metrics (e.g. Hamming distance) could be appropriate for other types of problems. After the cosine similarity is computed between the query HV and each class HV in the classifier, the class with the highest cosine similarity is chosen as the output class.

D. Challenges

There are challenges when running HD computing on embedded devices with limited resources. Storing HVs in $D = 1,000$ dimensions may require more resources than these devices have. Additionally, $k * D$ multiplications and additions need to be performed upon inference on a model with k classes and $D = 1,000$ dimension. This is costly for embedded devices with limited resources.

One solution is to reduce the dimensionality of the HD model. This method is effective at reducing the model size as well as the number of operations for inference [7]. However, as Table I shows, lowering dimensionality results in a trade off between accuracy and efficiency. As the dimensionality reduces, efficiency increases, i.e., faster and more energy-efficient computation, at the cost of accuracy. For example, when D is reduced from 1,000 to 250, on average, there is a 2.78× speed up at the cost of 39.23% of classification

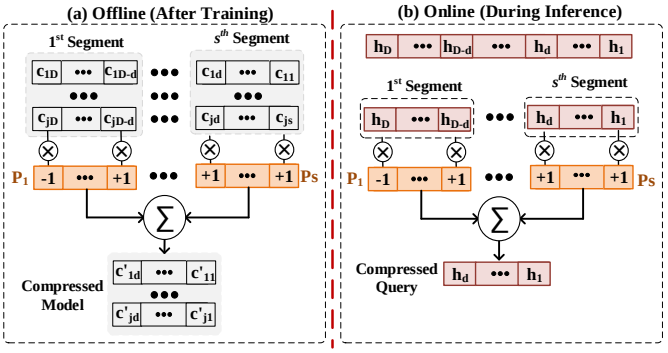
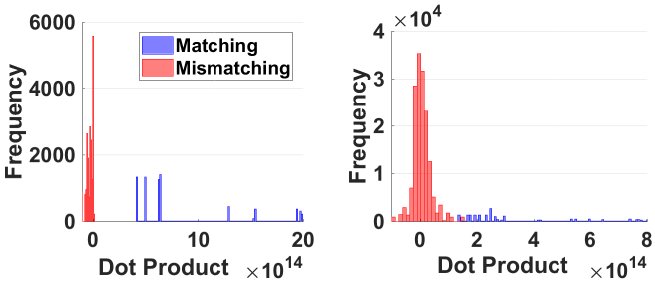


Fig. 2. CompHD compression of (a) an HD model and (b) a query data.



(a) s = 4 (b) s = 10

Fig. 3. Histogram of the distributions of the dot products of matching terms (data) and mismaching terms (noise)

accuracy. Therefore, simply reducing dimensionality does not provide acceleration without the cost of losing accuracy. Our goal is to design a framework which enables dimension reduction in HD computing with no or minimal impact on the classification accuracy.

III. Model Compression

Here we present our novel approach to accelerate HD computing by reducing the dimensionality. CompHD exploits the mathematics of high-dimensional spaces in order to reduce the effective dimensionality of the trained HD model while providing minimal loss in accuracy. Instead of using vectors with $D = 1,000$ dimensionality representing each class HV, CompHD compresses each class HV to d dimensionality where $d << D$. CompHD splits the trained class HVs into s equal segments, where each piece has $d = D/s$ dimensions. We then combine all s segments of each class HV to create a new HV in d dimensions. Combining these segments needs to preserve the information of each individual partition, otherwise CompHD would lose classification accuracy.

A. Compression

Each class HV is split into s equal segments with $d = D/s$ dimensionality, $\mathcal{C}_i = \{\mathcal{C}_i^1, \mathcal{C}_i^2, \ldots, \mathcal{C}_i^s\}$, where $\mathcal{C}_i^j$ is the j^{th} piece of the i^{th} class HV. CompHD could generate a d dimension class HV by just adding these segments together. However, this approach does not keep the positional information of each individual piece, which is important since HD works based on the pattern of similarity in high dimensional space. Thus, it is crucial to know the pattern of each individual partition.

To preserve the positional information, CompHD generates a set of HVs, $\{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_s\}$, where $\mathcal{P}_i \in \{-1, 1\}^D$. These HVs are generated semi-randomly with the Hadamard method [20] to ensure that they are mutually orthogonal. We do this by generating a $d \times d$ sized Hadamard matrix, which is a matrix with elements of $+1$ and -1 where each row is mutually orthogonal to every other row. We only require the use of the first s rows to use as our set of HVs, $\mathcal{P}$, because we only need one $\mathcal{P}$ for each segment. Using these HVs, we can uniquely store the information of each partition in a combined HV. We compress the class HVs by multiplying each segment of the class HV by a unique $\mathcal{P}$ and then adding each result up to create the compressed HV. The following equation shows how this compression is calculated:

$$\mathcal{C}' = \sum_{i=1}^{s} \mathcal{P}_i \mathcal{C}^i$$

Figure 2 (a) shows how CompHD creates a compressed HD model using trained class HVs and a set of $\mathcal{P}s$. This approach reduces the HD model dimension from D to d, where $d << D$.

B. Inference

To match the dimensionality of the HD model, the query hypervector is compressed using the same procedure as used for the class HVs.

$$\mathcal{Q}' = \sum_{i=1}^{s} \mathcal{P}_i \mathcal{Q}^i$$

Figure 2 (b) shows how CompHD compresses the query HV during inference. In testing, HD computing checks the similarity of the query HV with each compressed class HVs by calculating the cosine similarity: $(\mathcal{C} \cdot \mathcal{Q})/(\|\mathcal{C}\| \, \|\mathcal{Q}\|)$. Cosine similarity can be simplified to just calculating the dot product plus a division by storing the length of the class hypervectors. The division by the length of the query hypervector can be dropped because it simply scales the result and does not change which class will be selected. Using the non-compressed model, HD can perform the dot product between the query and class HV in D dimensions.

$$\delta = \frac{\mathcal{C} \cdot \mathcal{Q}}{\|\mathcal{C}\|}$$

This operation is very costly for $D = 1,000$ because it takes D multiplications and additions. After compressing the model to $d = D/s$ dimensionality, calculating the dot product can be done in d dimensions. Thus, the compressed model gains an approximate speed up of s over the full model upon inference. The speedup is approximately s because the query HV needs to be compressed as well before calculating the cosine similarity with the compressed model.

Once the query HV is compressed, CompHD selects the class with the highest cosine similarity to the compressed query HV, which is calculated using:

$$argmax_{i=1:k}\{\delta\langle \mathcal{Q}', \mathcal{C}'_i\rangle\}$$

To maintain the accuracy of the full sized HD model, the dot product between the compressed class HV (C'_i) and compressed query HV (Q') needs to be as close to the dot product of the full sized class HV (C_i) and full sized query HV (Q). When the dot product of the compressed model is foiled out, the terms of the resulting dot product of the compressed model can be split up into two parts, noise and data. Noise occurs when a term has mismatching $\mathcal{P}s$ and data occurs when a term has matching $\mathcal{P}s$.

$$\mathcal{Q}' \cdot \mathcal{C}'_i = (\sum_{i=1}^{s} \mathcal{P}_i \mathcal{C}^i) \cdot (\sum_{j=1}^{s} \mathcal{P}_j \mathcal{Q}^j)$$

$$\mathcal{Q}' \cdot \mathcal{C}'_i = \underbrace{\sum_{i=j} \mathcal{P}_i \mathcal{C}^i \mathcal{P}_i \mathcal{Q}^i}_{\text{data}} + \underbrace{\sum_{i\neq j} \mathcal{P}_i \mathcal{C}^i \mathcal{P}_j \mathcal{Q}^j}_{\text{noise}}$$

If we only kept the terms with matching $\mathcal{P}s$, the resulting dot product would be equal to the dot product of the full sized model. Therefore, to achieve the same results as the full sized model our design needs to minimize the noisy terms.

CompHD achieves this by ensuring that every $\mathcal{P}$ is mutually orthogonal to each other by generating them with the Hadamard method. Therefore, when two different $\mathcal{P}s$ are in the same term, their inner product is approximately zero and the resulting dot product is minimized. Thus, resulting in the dot products of the compressed model being approximately equal to the dot products of the full sized model. Figure 3 shows the distributions of the dot products of data terms and the dot products of noisy terms for the Valve Monitoring dataset with s = 4 and s = 10. It is clear that data terms are contributing to the resulting dot product significantly more than noisy terms. Therefore, the error introduced by noisy terms does not have much impact on the resulting dot product. This leaves the dot product between C'_i and Q' with the the terms where there are matching $\mathcal{P}s$, because all other terms are minuscule in comparison:

$$\mathcal{C}' \cdot \mathcal{Q}' \approx \sum_{i=j} \mathcal{P}_i \mathcal{C}^i \mathcal{P}_i \mathcal{Q}^i$$

This is the desired result because the dot product in the compressed model is approximately equal to the dot product of the full sized model, thus reducing the error introduced when compressing the model. Additionally, we can see in figure 3 when the compression factor increases, the distance between noisy terms and data terms is reduced, resulting in a less accurate model. CompHD uses this information about the ratio of noisy data to real data to selectively pick the best compression factor to use that gives comparable accuracy to the full sized model with a significant improvement on efficiency.

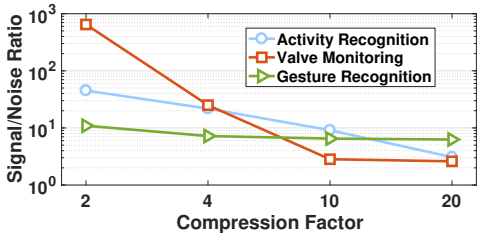


Fig. 4. Ratio of real data to noisy data for different values of effective dimension

IV. Evaluation

A. Experimental Setup

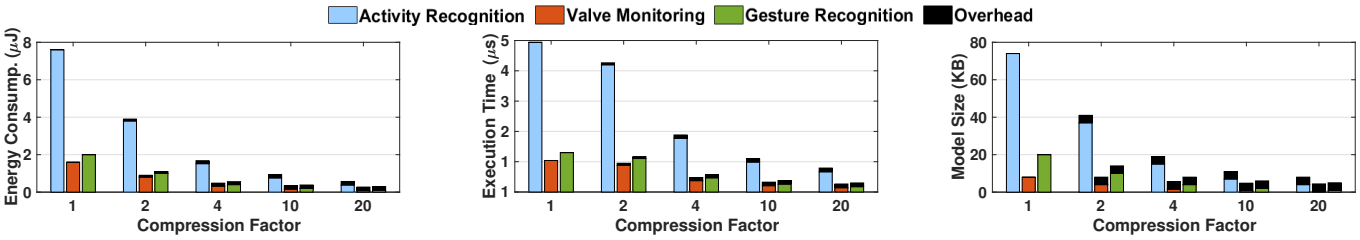
We implemented CompHD training and inference in both software and hardware. In software, we implemented CompHD using C++ code. We also implemented CompHD on two embedded devices: a Raspberry Pi 3 using ARM Cortex A53 CPU and a Kintex-7 FPGA. For the FPGA, we implemented CompHD using Verilog. We verify the timing and the functionality of the models by synthesizing them using the Xilinx Vivado Design Suite [21]. The synthesis code has been implemented on the Kintex-7 FPGA KC705 Evaluation Kit.

We test the efficiency of the proposed approach on three practical applications:

Activity Recognition [22]: Using motion sensor data from 5 sensor units with each unit containing 9 sensors, the objective is to recognize the activity performed. The training and testing datasets are taken from the Daily Sports and Activities dataset. This dataset consists of 8 subjects performing 19 different activities in their own style for a 5 minute duration. All of the sensors record data at 25Hz during the 5 minute interval and each 5 minute interval is then divided into 5 second intervals to create 60 separate data samples each containing 5,625 data points.

Valve Monitoring [23] The goal of this task is to determine if the condition of the valves in a hydraulic system are optimal, have a small lag, have sever lag, or are failing. The training and testing datasets are taken from the Condition Monitoring of Hydraulic Systems dataset. This dataset consists of 2205 samples of sensor data from a hydraulic system. Each sample has the data from 17 separate sensors over a duration of 60 seconds totaling 43,680 data points per sample.

Gesture Recognition [24]: Here we try to recognize five different hand gestures: rested hand, closed hand, open hand, 2-finger pinch, and point index. The gestures were sampled at 500Hz with the use of an elastic band containing four EMG sensors. We used the data collected from five different subjects. The data was collected by each subject performing 10 repetitions of each gesture for three seconds each with a three second resting period in between. Therefore, each sample contains 6,000 data points.



(a) (b) (c)

Fig. 5. Energy consumption, execution time, and model size of CompHD using different compression factors.

TABLE II
COMPARING THE EFFECT OF REDUCING MODEL SIZE WITH CompHD AND DIMENSION REDUCTION ON ACCURACY

Dataset		Activity Recognition		Valve Monitoring		Gesture Recognition	
Effective D	s	Baseline	CompHD	Baseline	CompHD	Baseline	CompHD
1,000	1	100%	100%	100%	100%	91.04%	91.04%
500	2	100%	100%	100%	100%	88.33%	90.66%
100	10	5.26%	100%	51.02%	100%	62.88%	91.17%
50	20	5.26%	100%	50.32%	100%	39.01%	89.94%
25	40	5.26%	57.89%	51.02%	83.67%	26.85%	86.69%

B. CompHD & Compression Factor

Figure 4 shows the ratios of real data in the dot product of the compressed model to the noisy data in the dot product of the compressed model. The graphs show that when increasing the compression factor, the ratio of real data to noisy data decreases. This is because as the compression factor increases, the amount of real terms in the dot product linearly increases. Meanwhile, the amount of noisy terms quadraticly increases. Therefore, even though the noisy terms are much smaller than the real data terms, as the compression factor increases, the amount of noisy terms grows faster than the amount of real data terms, bringing down the ratio. This results in a decrease in accuracy as the compression factor increases too far. CompHD chooses a compression factor such that the ratio of real data to noisy data in the dot product is sufficiently high to maintain a comparable accuracy to the full sized model and the compression factor is large enough to improve on efficiency over the full sized model. Based on our results, a data ratio of 5 or higher is enough to ensure a comparable accuracy to the full sized model. As figure 4 shows, CompHD chose a compression factor of s = 20 for all three datasets.

C. CompHD Accuracy

Table II compares the classification accuracy of CompHD with the classification accuracy of the baseline method dimension reduction as the model size decreases. The data shows that when reducing the length of the hypervectors with dimension reduction, there is a significant trade off between model size and classification accuracy. For example, on the Valve Monitoring dataset, when dimension reduction reduces the length of the hypervectors by 90%, the model loses 48.98% accuracy. As the dimensionality is reduced further with dimension reduction, more accuracy loss is observed. CompHD reduces this trade off by a significant amount by compressing the full sized model rather than simply reducing the dimensionality.

CompHD is able to reduce the model size while maintaining a comparable accuracy to a full sized model. For instance, when reducing the length of the hypervectors by 95% with CompHD, the model maintains the same accuracy as the original model for the Valve Monitoring dataset. On average, CompHD loses 65.33% less accuracy than dimension reduction while reducing the length of the hypervectors by 95%. Although, there is a point where CompHD loses more accuracy than desired for each dataset. For example, with the Gesture Recognition dataset, when $s = 40$ the accuracy drops by 4.35% from the original model. However, CompHD is still 27.21% more accurate than dimension reduction. This shows that CompHD is capable of maintaining the accuracy of the original model while reducing the model size up to a break point. Additionally, CompHD is strictly better than dimension reduction at reducing model size because CompHD never loses as much accuracy as dimension reduction for the same model size reduction. Our evaluation shows that CompHD is a robust way to reduce model size while maintaining a comparable accuracy to the original model.

D. CompHD Efficiency

Figure 5 compares the energy consumption, execution time, and model size of CompHD using different compression factors. The baseline is also represented in the graphs, as a compression factor of 1 is the baseline. The data shows that CompHD improves the energy consumption, execution time, and model size of HD computing as the compression factor increases. All results are reported when applications are running on a Kintex-7 FPGA. As stated before, the improvement is closely linear with respect to the compression factor. For example, when s = 10 for the Activity Recognition dataset, CompHD uses $8.09\times$ less energy than the baseline and gains a speed up of $4.47\times$. However, because of the overheads of our design, it is not completely linear. The overheads of our design come from the need to compress hypervectors from the original high-dimensional model. Compressing the query HV may seem expensive, however, due to $\mathcal{P}s \in \{-1, 1\}^D$, the multiplications with the class HV segments can be reduced to deciding if the subsequent operation when combining the segments will be addition or subtraction. The graph of energy consumption shows the additional energy needed by CompHD

TABLE III
EFFICIENCY IMPROVEMENT AND SPEEDUP OF CompHD OVER DIMENSION REDUCTION FOR THE SAME ACCURACY.

Dataset	Activity Recognition	Valve Monitoring	Gesture Recognition
Energy Improv.	85%	66.25%	71%
Speedup	5.31×	3.35×	3.7×

to compress the query hypervector. Additionally, the graph of execution time shows the additional time needed to compress the query hypervector. Lastly, the graph showing the model size of CompHD shows how much additional space is need for the $\mathcal{P}s$ that are needed to compress the query hypervector. Despite these overheads, the graphs in figure 5 show that CompHD still improves the energy consumption, execution time, and model size of HD computing nearly linearly with respect to the compression factor.

E. Efficiency Considering Quality

Table III compares the efficiency of CompHD with dimension reduction when they provide the same accuracy. The dimension reduction design and CompHD are compared by implementing them on the Kintex-7 FPGA KC705 Evaluation Kit. The data highlights the improvement that CompHD has over dimension reduction at the same accuracy. CompHD is able to provide the same accuracy as dimension reduction while saving more energy. This is because dimension reduction improves the efficiency of the HD model by just lowering the dimensionality. This reduces the amount of information that can be stored in the HD model, causing a significant loss of accuracy. CompHD reduces this trade off by compressing the model instead of just lowering the dimensionality. Compressing the HD model saves important information in the larger model needed to keep the accuracy high. This allows CompHD to decrease the model size further than dimension reduction while providing the same accuracy. Even though CompHD is less efficient than dimension reduction at the same effective dimension due to the need of compressing the query HV, by preserving the information of the larger model, CompHD is able to reduce the dimensionality even further. Overall, CompHD is able to speed up execution time by an average of $4.1\times$ and improve efficiency by an average of 74% more than dimension reduction while providing the same accuracy.

V. CONCLUSION

In this paper, we proposed a new method to reduce the size of HD models without a trade-off of accuracy. CompHD achieves this by dividing the class HVs into s segments and combining those segments together with well defined vector operations that reduce the amount of information lost. Once combined, the new class HVs have dimensionality $d = D/s$. This speeds up inference by approximately s times and reduces the model size by approximately s times. This enables HD to be run on a wider range of embedded devices with limited resources. CompHD can reduce model size by an average of 69.7%, resulting in a execution time speeding up by $4.1\times$ and improving energy efficiency by 74% while maintaining the same accuracy of the original model. Our results show that CompHD enables more low powered IoT devices to solve ML problems with HD computing.

ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[4] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[5] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[6] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2015.

[7] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *Proceedings of the 55th Annual Design Automation Conference*, p. 108, ACM, 2018.

[8] M. Imani *et al.*, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *DATE*, IEEE/ACM, 2019.

[9] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *IEEE CLOUD*, pp. 1–6, IEEE, 2019.

[10] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *International Conference on Rebooting Computing (ICRC)*, pp. 1–6, IEEE, 2017.

[11] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 445–456, IEEE, 2017.

[12] M. Imani *et al.*, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *ASPDAC*, pp. 493–498, ACM, 2019.

[13] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, pp. 53–62, ACM, 2019.

[14] S. Gupta *et al.*, "Felix: fast and energy-efficient logic in memory," in *ICCAD*, p. 55, ACM, 2018.

[15] M. Imani *et al.*, "A binary learning framework for hyperdimensional computing," in *DATE*, IEEE/ACM, 2019.

[16] M. Imani *et al.*, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *IEEE FCCM*, pp. 1–6, IEEE, 2019.

[17] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[18] P. Kanerva, "Encoding structure in boolean space," in *ICANN 98*, pp. 387–392, Springer, 1998.

[19] A. Rahimi, P. Kanerva, J. d. R. Millán, and J. M. Rabaey, "Hyperdimensional computing for noninvasive brain-computer interfaces: Blind and one-shot classification of eeg error-related potentials," in *10th EAI Int. Conf. on Bio-inspired Information and Communications Technologies*, 2017.

[20] "Hadamard matrix." https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.linalg.hadamard.html.

[21] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.

[22] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities.

[23] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/Condition+monitoring+of+hydraulic+systems.

[24] S. Benatti, E. Farella, E. Gruppioni, and L. Benini, "Analysis of robust implementation of an emg pattern recognition based control.," in *BIOSIGNALS*, pp. 45–54, 2014.