# Multi-label HD Classification in 3D Flash

Justin Morris*†, Yilun Hao*, Saransh Gupta*, Ranganathan Ramkumar*, Jeffrey Yu*, Mohsen Imani‡,
Baris Aksanli†, and Tajana Rosing*
*University of California San Diego, La Jolla, CA 92093, USA
†San Diego State University, San Diego, CA 92182, USA
‡University of California Irvine, Irvine, CA 92697, USA
{justinmorris, yih301, sgupta, rramkuma, jey070}@ucsd.edu, m.imani@uci.edu, baksanli@sdsu.edu, tajana@ucsd.edu

***Abstract*—Many classification problems in practice map each sample to more than one label - this is known as multi-label classification. In this work, we present Multi-label HD, an in 3D storage multi-label classification system that uses Hyperdimensional Computing (HD). Multi-label HD is the first HD system to support multi-label classification. We propose two different mappings of HD to Multi-label HD. The first, Power Set HD, transforms the multi-label problem into single-label classification by creating a new class for each label combination. The second, Multi-Model HD, creates a binary classification model for each possible label. Our evaluation shows that Multi-Model HD achieves, on average, $47.8\times$ higher energy efficiency and $47.1\times$ faster execution time while achieving $5\%$ higher classification accuracy as state-of-the-art light-weight multi-label classifiers. Power Set HD achieves $13\%$ higher accuracy than Multi-Model HD, but is $2\times$ slower. Our 3D-flash acceleration further improves the energy efficiency of Multi-label HD training by $228\times$ and reduces the latency by $610\times$ vs training on a CPU.**

## I. Introduction

The emergence of the Internet of Things (IoT) has created an abundance of small embedded devices [1]. Many of these devices perform classification tasks, such as speech recognition, image classification, etc. More and more devices are now required to perform more complex multi-label classification [2], [3]. However, they have very limited resources such as limited battery lifetime and a small amount of memory, which is not enough to train and run Deep Neural Networks (DNN) [4]. We need a light-weight classification algorithm to perform such tasks on embedded systems.

Brain-inspired Hyperdimensional (HD) computing has been proposed as the alternative light-weight computing method to perform cognitive tasks on devices with limited resources [5]. Inspired by the pattern of neural activity in the brain [6], HD computing maps each data point into high dimension vectors, called *hypervectors* (HVs). HD computing has three main stages, 1) Encoding: mapping data into HVs. 2) Training: combining encoded HVs to create a model representing each class with a HV. 3) Inference: comparing the incoming sample with the trained model to find the most similar class. HD computing shows promising progress for many cognitive tasks such as activity recognition, object recognition, language recognition, and bio-signal classification [7], [8], [9]. However, there has been no work yet on mapping HD computing to multi-label classification tasks.

978-1-7281-5409-1/20/$31.00 ©2020 IEEE

While HD provides improvements in performance and energy consumption over conventional machine learning algorithms, it still involves fetching each and every data from memory/disk and processing it on CPUs/GPUs. This is exaserbated by the fact that HD expands the dimensionality of the input data into high dimensional space. This massive amount of data needed for HD cannot always fit into the memory. Recent work has introduced computing capabilities to solid-state disks (SSDs) to process data in storage [10], [11], [12], [13], [14]. This not only reduces the computation load from the processing cores but also processes raw data where it is stored. HD computing has compelling properties for efficient hardware acceleration in flash. For instance, HD is highly parallelizable with $D = 10,000$ dimensions where each dimension is independent. Furthermore, HD is comprised of simple operations such as addition, multiplication, and comparisons. With these two properties, HD computing is a prime candidate for acceleration in flash.

In this paper, we design a new Multi-label HD computing in storage system. Our system efficiently accelerates the data-intensive steps of HD, encoding and training, in 3D storage, thus, making it possible to run multi-label classification with HD in the IoT domain. We propose two different mappings of HD to multi-label classification, Power Set HD and Multi-Model HD. Power Set HD, transforms the multi-label problem into classical classification by creating a new class for each label combination. Multi-Model HD that creates a binary classification model for each possible label. Our evaluation shows that Multi-Model HD achieves, on average, $47.8\times$ higher energy efficiency and $47.1\times$ faster execution time while achieving $5\%$ higher classification accuracy as state-of-the-art light-weight multi-label classifiers such as multi-label kNNs. Power Set HD achieves $13\%$ higher accuracy than Multi-Model HD, but is $2\times$ slower. Using our in-3D-flash acceleration, we further improve the energy efficiency of Multi-label HD training by $228\times$ and reduce the latency by $610\times$.

## II. Related Work

### A. Hyperdimensional Computing

Prior work tried to apply the idea of high-dimensional computing to different classification problems such as language recognition, speech recognition, face detection, EMG gesture detection, human-computer interaction, and sensor

fusion prediction [7], [15], [16], [17]. Additionally, work in [9] proposed a new HD encoding based on random indexing for recognizing a text's language by generating and comparing text hypervectors. Work in [18] proposed an encoding method to map and classify biosignal sensory data in high dimensional space. Work in [8] proposed a general encoding module that maps feature vectors into high-dimensional space while keeping most of the original data. There is no work to date that handles multi-label classification in HD.

### B. Multi-label Classification

Prior work applied problem transformation methods to transform multi-label classification problems into multiple single-label classification problems [2], [3]. The most widely used transformation method is PT3. PT3 combines each different set of labels into a single label so that the new label set $L^{'}$ is the power set of the old label set $L$. For a dataset with three binary labels, the new label set would be 000, 001, 010, 011, 100, 101, 110, 111. This causes an exponential increase in the number of labels in the dataset. This transformation method is popular for other light-weight classifiers as their complexities mostly scale with the number of features and not with the number of labels. However, in HD computing, the complexity of inference does scale with the number of labels. Therefore, in this paper we propose a new Multi-Model transformation method that is designed for scalable HD computing.

### C. Hardware Acceleration

**HD Acceleration on other Platforms:** Prior work tried to design different hardware accelerators for HD computing. This includes accelerating HD computing on existing FPGA, ASIC, and processing in-memory platforms [19]. However, these solutions do not scale well with the number of classes and dimensions, primarily due to the data movement issue. Therefore, a new solution is needed that can scale with the dimensionality and number of classes. ISC is a promising acceleration architecture in this aspect. **Computing in 3D Flash:** The current 3D flash-based storage systems suffer from slow flash array read latency and storage to host I/O latency. To alleviate these issues prior work introduced in-storage computing (ISC) architectures [12]. These works exploit the embedded cores present in the SSD controller to implement ISC. Another set of work in [11], [20] used ASIC accelerators in SSDs. The work in [13] proposed a full-stack storage system to reduce the host-side I/O stack latency. While these works propose single-level computing in storage, [14] on the other hand exploited computing at flash die and in top level accelerator to provide multi-layer computing. It also allows for high parallelism in computation. In this work, we adapt the ISC design in [14] to enable multi-label HD in 3D flash.

## III. Multi-label Classification with HD

Multi-label classification is the problem of finding a model that maps inputs $x$ to binary vectors $y$, and each element in $y$ is a label that is assigned a value either 0 or 1. This is in contrast to single-label classification, where y is a single value, not a vector of labels. Although HD computing performs well for single-label classification tasks, we can't directly apply it to solve multi-label classification problems, as only one label output is chosen. Therefore, we transform the multi-label problem into a single-label problem and then modify the HD computing algorithm to solve the multi-label classification problem. We propose two different mappings of HD to Multi-label HD. The first, Power Set HD, transforms the multi-label problem into single-label classification by creating a new class for each observed label combination. We additionally propose Multi-Model HD that creates a binary classification model for each possible label. By doing this, we can leverage the efficiency of HD computing to complete the multi-label classification task faster and with less energy consumption.

### A. Problem Transformation Methods

**Power Set:** Prior work[2] mapped multi-label classification to single label classification by creating a new label set that was the power set of the multi-labels. For instance, if a multi-label problem had 3 possible labels for every sample, then prior work would transform the 3 multi-labels into 8 single labels. Where each single label represents each possible combination of the 3 individual labels. This exponential increase in the number of labels does not cause challenges for classifiers that do not scale in complexity with the number of labels. However, HD computing complexity does scale with the number of labels. We address this issue with a binary classification transformation for HD computing explained below.

**Multi-Model:** We propose Multi-Model HD, a method of building a binary classification model for each label as the problem transformation method. Suppose $[l_1...l_h]$ are the labels of the dataset, then after mapping each data point into hypervectors $[v_1...v_n]$, we build $h$ binary classification models, since each label only has a true or false value, i.e., 0 or 1. For example, if a dataset has $h = 3$, we create 3 different HD models, one for each label. Then upon inference, we feed the input data into all 3 of the models, independently checking for the existence of each label. This transformation method is better for HD in multiple ways: 1) HD model size, execution time, and energy scale with the number of classes, so when using Power Set HD, if there is a large number of possible label combinations, Power Set HD will not be as efficient as Multi-Model HD. 2) If a new label is introduced, in Multi-Model HD, we simply need to train a newly added binary classification HD model. However, with Power Set HD, or other models that use the power set transformation method, the entire model needs to be retrained to accommodate the new label combinations. The rest of Section III is mainly focused on Multi-Model HD, while we additionally provide a comparison with Power Set HD in Section V. Now that the problem has been transformed into $k$ binary classification problems, we describe the algorithmic changes to HD computing blow.

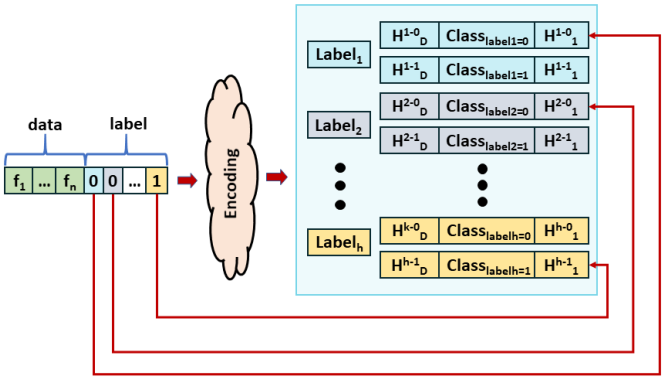



Fig. 1. An example of how the Multi-Model HD model is created

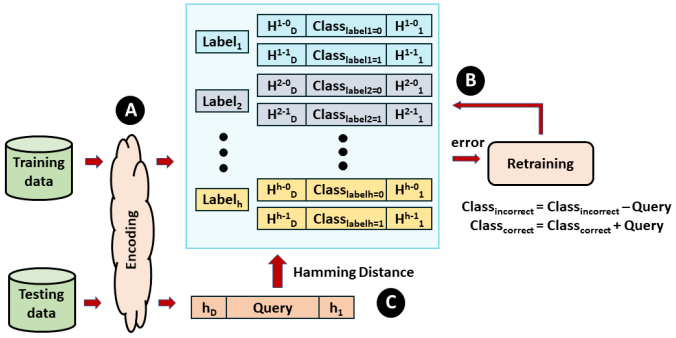



Fig. 2. Overview of how Multi-Model HD is constructed and how Multi-Model HD performs inference.

### B. *Encoding*

HD computing encoding maps each $n$ dimensional feature vector to a $D$ dimensional binary hypervector. We utilize a random projection encoding presented in [21]. 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 encoding is to map this feature vector to a $D$ (e.g. $D = 10,000$) dimensional space vector: $\mathbf{H} = \{h_1,\ h_2, \ldots,\ h_D\}$. The encoding first generates $D$ dense bipolar vectors with the same dimensionality as original domain, $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_D\}$, where $\mathbf{p}_i \in \{-1, 1\}^n$. Thus, to encode a feature vector into a hypervector, we perform a matrix vector multiplication between the projection matrix and the feature vector using:

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

.

Where $sign$ is a sign function which maps the result of the dot product to +1 or -1. In Section IV-A we discuss how we accelerate encoding in flash.

### C. *Training*

In HD computing, the model used in training is initialized through element-wise addition of all encoded hypervectors in each existing class. The result of training are $k$ hypervectors each with $D$ dimensions, where $k$ is the number of classes. For example, the $i^{th}$ class hypervector can be computed as: $\mathbf{C_i} = \sum_{\forall j \in class_i} \mathbf{T_j}$. However, to map this algorithm to Multi-label classification, we need to modify how we create the initial HVs.

As stated in Section III-A, since the multi-label classification problem is transformed into multiple binary classification problems with Multi-Model HD, we build two classes for each label (one for value 0 and one for value 1). As shown in Figure 1, after the same encoding process as stated in Section III-B, each data point is classified into either $Class_{label_i=0}$ or $Class_{labeli=1}$ for each label $i$ according to the values of its labels $[l_1 ... l_h]$. As shown in Figure 2 ❹, for a dataset that has $h$ labels, the binary model of this dataset would contain $2h$ class HVs in total, one binary classification model for each label where each model contains 2 class HVs.

Unlike in single label classification, in Multi-Model HD, each data point is element wise added to multiple class HVs. For instance, in Figure 1, after the sample is encoded, it is added to the $Class_{label_1=0}$ class HV for the first label, as the first label is 0. It is then additionally added to the $Class_{label_2=0}$ class HV for the second label, as the second label is 0. This is continued for all the labels until it is added to the $Class_{label_h=1}$ class HV for the last label, as the last label is 1. After this procedure is repeated for the entire training set, we are left with $k$ classification models for each label.

This training procedure also results in integer values for the dimensions of the class HVs, requiring the use of a costly cosine similarity during inference to find the best matching class HV to the query HV. We can reduce this computation to a binary operation of Hamming distance by binarizing the model, which is done by changing the class hypervector elements to +1 if they are positive and -1 if they are negative or 0. Hamming distance is desirable because it reduces each multiplication and addition in cosine similarity to a simple bitwise XOR and accumulation, which is significantly more efficient in acceleration circuits. The class with the least mismatching bits to the query is then chosen as the output.

### D. *Inference*

After training, the HD model for single-label classification can now be used for inference. Upon inference, an input data is encoded to a *query* hypervector using the same encoding module used for training. HD Computing then computes the similarity between the *query* hypervector and each class hypervector. It then uses consine similarity to find a class hypervector with the most similarity with the query hypervector.

Multi-Model HD performs inference in a similar way, however, we need to output $h$ labels instead of just 1. Figure 2 ❸ shows how inference is performed in Muti-Model HD. Upon inference, an input data is encoded to a *query* hypervector using the same encoding module used for training, just like baseline HD. However, since Multi-Model HD contains $h$ different classification models, the query HV is input into

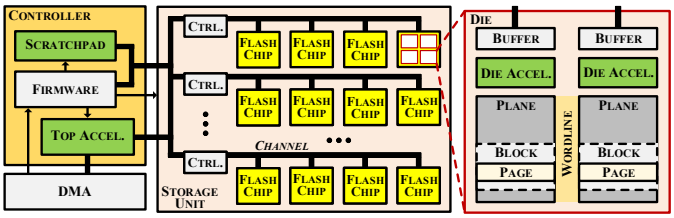



Fig. 3. Overview of Multi-label HD in 3D flash-based storage. ISC enabling components of the design are shown in green.

each classification model independently. For each model, if the query HV is more similar to the 0 label HV, then that label output is chosen as 0, and vice versa if the query is more similar to the 1 label HV. This generates our $h$ different labels for output in a multi-label classification problem. In Multi-label HD, inference is performed on the host CPU.

## IV. Acceleration with 3D NAND Flash

Here, we present an ISC design that performs Multi-label HD encoding and training completely in 3D flash. Figure 3 shows an overview of the SSD architecture we adopt from THRIFTY [14]. It uses a die-level accelerator (green on the right in Figure 3), in each plane to encode every read page into a hypervector. These hypervectors are then sent to a SSD-level FPGA, which accumulates the hypervectors in the top-level accelerator (green on bottom left in Figure 3) to perform training. The scratchpad (green on top left in Figure 3) in the controller stores the encoding projection matrix, which it receives as an application parameter from the host. The top-level accelerator is an FPGA which uses INSIDER acceleration cluster [13] to implement HDC accumulation and other operations. We utilize THRIFTY's adaptation of INSIDER's software stack to connect our ISC architecture to the rest of the system.

### A. Encoding in 3D Flash

As shown in Figure 3, the flash chip may consist of several flash dies which are further divided into flash planes, each plane consisting of a group of blocks, each of which store multiple pages. Each plane has a page buffer to write the data to. Operations in SSD happen in page granularity where the size of the pages usually ranges from 2KB-16KB. Hence, we use accelerators for each flash plane to exploit the flash hierarchy. These accelerators are multiplexed to the page read path.

The die-accelerator in [14] encodes an entire page with raw data to generate a $D$ dimensional hypervector. We assume that the feature vectors are page-aligned, with each page storing one full feature vector. Multi-label HD encoding multiplies an $n$-size feature vector with a projection matrix containing $D \times n$ 1-bit elements. The accelerator calculates the dot product between the two vectors, one read from the flash array and another being a row-vector of the projection matrix. This involves element-wise multiplication of the two vectors and adding together all the elements in the product. Since the weights in the projection matrix $\in \{1, -1\}$, we map them to $\{0, 1\}$ respectively. We use 2's complement to break the multiplication into an inversion using XOR gates and then add the total number of inverted inputs to the accumulated sum of XOR outputs. With the assumption that each page consists of a maximum 1K feature elements, the accelerator consists of an array of 32K XOR gates followed by a 1024 input tree adder. It reduces 1024 inputs to 2, which is followed by a carry look ahead addition to get the final dot product. The sign bit (MSB) of the output is the value of one dimension of the encoded hypervector. Complementary to the projection matrix, the output $0 \rightarrow 1$ and $1 \rightarrow (-1)$. The accelerator is iteratively run $D$ times to generate $D$ dimensions. Each encoded hypervector is appended with the corresponding label vector. We write the output of the accelerator to the page buffer of the plane, which serves as the response to the original read request.

### B. Training at Top-Level in Storage

The encoded hypervectors from flash chips are input into the top-level accelerator, which is implemented on an FPGA present in the SSD. During training, they are accumulated into the corresponding label hypervectors. At the end of training we obtain two output hypervector for every label ($label_i$), one each for $Class_{label_i=0}$ and $Class_{label_i=1}$.

The design in [14] utilized input queues for each class to increase parallelism between different classes. However in Multi-label HD, each encoded hypervector is added to one of the two classes of each label, i.e. 50% of the classes. Moreover, ideally an encoded hypervector has just one label as '1' while rest are '0's. Hence, all but one classes corresponding to $label_i = 0$ would receive an incoming hypervector. There is negligible parallelism in training between multiple encoded hypervectors. In this case, the input queues of [14] are an overkill. Hence, we remove input queues from the FPGA design of [14]. The label vector of an incoming hypervector is used to input it to the corresponding class ($Class_{label_i=0}$ or $Class_{label_i=1}$) of each label. The inputs to the remaining classes are set to zero. An accumulator is present for each class, which simply needs to read the input and operate on the corresponding data. The accumulators for each class operates in parallel to add an input hypervector to the corresponding class hypervector. While the computation can also be fully parallelized over all dimensions, the large size of hypervectors and the limited read ports of the memory make it impractical. Hence, we utilize the partition-based approach used in [14] to allow partial parallelism. The final class hypervectors are sent to the host.

## V. Experimental Results

### A. Experiment Setup

We tested Multi-label HD training and inference using an optimized C++ implementation. For comparison, we utilized the open source Mulan multi-label package, which is implemented in Java [22]. We compare Multi-label HD with multi-label versions of k-nearest neighbors (kNN), Sequential

TABLE I
MULTI-LABEL HD 3D STORAGE PARAMETERS

| | | | |
|---|---|---|---|
| **Capacity** | $1TB$ | **Channels** | 32 |
| **Page Size** | $16KB$ | **Chips/Channel** | 4 |
| **External BW** | $3.2GBps$ | **Planes/Chip** | 8 |
| **BW/Channel** | $800MBps$ | **Blocks/Plane** | 512 |
| **Flash Latency** | $53us$ | **Pages/Block** | 128 |
| **FPGA** | $XCKU025$ | **Scratchpad Size** | $4MB$ |
| **Avg Power/DA** | $8mW$ | **DA Latency** | $1.02ns$ |

*DA: Die-accelerator

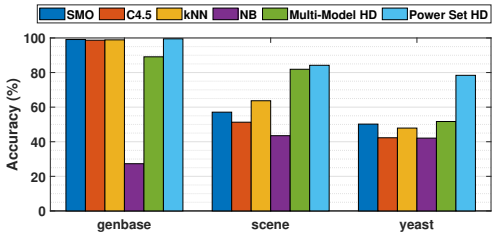
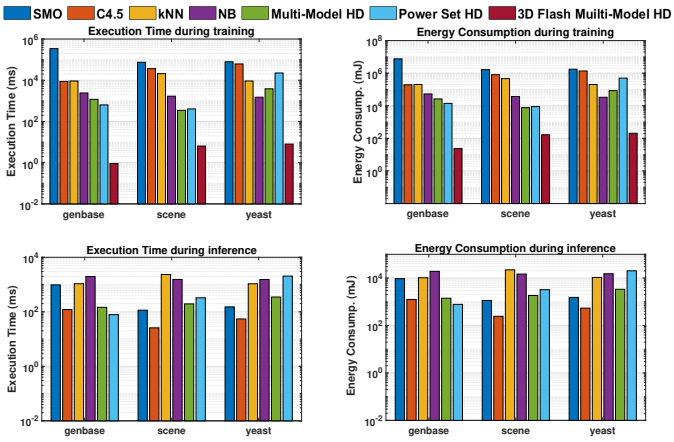



Fig. 4. Classification accuracy of Multilabel HD and other multi-label classification algorithms.




(a) Execution Time (b) Energy

Fig. 5. Energy consumption and execution time of Multi-label HD during Encoding and Training.

minimal optimization (SMO), C4.5, and Naive Bayes (NB). We also developed a simulator for Multi-label HD in flash which supports parallel read and write accesses to the flash chips. We utilized Verilog and Synopsys *Design Compiler* to implement and synthesize the die-level accelerator at 45nm and scale it down to 22nm. The top-level FPGA accelerator has been synthesized and simulated in Xilinx Vivado. For drive simulation, we assume the characteristics similar to 1TB Intel DC P4500 PCIe-3.1 SSD connected to an Intel(R) Xeon(R) CPU E5-2640 v3 host. The parameters for our 3D flash implementation are shown in Table I. We compare flash implementation with 6th Gen 3.2GHz Sky Lake Intel Core i5-6300HQ CPU with 8GB of RAM and a 256 GB SSD.

We tested our proposed approach on three applications:

**Genbase** (Genbase) [23]: The protein classes considered are the 27 most important protein families. The training and testing datasets are taken from the Genbase dataset. This dataset consists of 662 samples, each with 1186 attributes.

**Scene** (Scene) [24]: This dataset contains characteristics about images and their classes. One image can belong to one or more classes. The training and testing datasets are taken from the Scene dataset. This dataset contains 2407 samples, each with 294 attributes.

**Yeast** (Yeast) [25]: This database contains information about a set of Yeast cells. The task is to determine the localization site of each cell. The training and testing datasets are taken from the Yeast dataset. This dataset consists of 2417 samples, each with 103 attributes.

### B. Multi-label HD Comparison with State-of-the-Art

*1) Accuracy:*

Figure 4 compares the multi-label classification accuracy of current state-of-the-art multi-label classifiers with Multi-label HD. The accuracy for multi-label is calculated by first getting the accuracy of the model on each label individually. Then to aggregate them, we average each label's accuracy together to get one overall accuracy number for each dataset. As the figure shows, Multi-label HD (Multi-Model HD and Power Set HD) are comparable in accuracy to state-of-the-art multi-label classifiers. In fact, Power Set HD is always better than the state-of-the-art on these three datasets. On the other hand, Multi-model is slightly less accurate than other multi-label classifiers on the genbase dataset by 10%. However, Multi-Model HD is able to achieve higher accuracy on the scene and yeast datasets. This could be attributed to mapping the data into HD space, offering better separability than in the low dimensional data. However, more theoretical analysis on HD Computing is necessary in order to understand why Multi-label HD is more accurate. Overall, on average, Multi-Model HD is 5% more accurate and Power Set HD is 14% more accurate than the highest accuracy state-of-the-art multi-label classifier.

Although Power Set HD achieves higher accuracy than Multi-Model HD, Figure 5 demonstrates that the improvement in accuracy comes at a significant cost in execution time and energy. This is because of the exponential increase in class HVs as discussed in Section III-A. As mentioned before, the exception is the genbase dataset because there is only a small subset of possible combinations that appear in the dataset. On the other hand, when there is a large portion of possible combinations in the dataset, Power Set HD is $3.6\times$ slower than Multi-Model HD. This offers a trade-off between execution time and energy efficiency vs accuracy. If an application requires the highest accuracy, Power Set HD should be used. However, if the key metric is execution time and energy efficiency, for a loss in accuracy compared to Power Set HD, but still comparable with other state-of-the-art multilabel classifiers, Multi-Model HD is the clear choice. If the dataset does not have a diverse combination of labels, such as in genbase, Power Set HD can potentially be more accurate and energy efficient compared to Multi-Model HD.

*2) CPU Execution Time and Energy:*

Figure 5 compares the execution time and energy consumption of state-of-the-art multi-label classifiers with Multi-label HD on CPU. The data demonstrates that both Multi-Model HD and Power Set HD training are significantly faster than most other multi-label classifiers. On average, Multi-label HD is $60.8\times$ faster and $61.8\times$ more energy efficient than other multi-label classifiers during training. The one exception is Naive Bayes on the yeast dataset, however, although Naive Bayes trains significantly faster than Multi-Model HD on the yeast dataset, Multi-Model HD is $8.6\times$ faster and $8.7\times$ more energy efficient than Naive Bayes during inference. Additionally, Power Set HD is only $3.5\times$ slower than Multi-Model HD on datasets with a large portion of label combinations.

Figure 5 also demonstrates that Multi-Model HD is also significantly faster than kNNs and Naive Bayes multi-label models during inference. Although Multi-Model HD is comparable in execution time and energy efficiency to SMO and C4.5 during inference, Multi-Model HD is $174.4\times(42.8\times)$ faster and $178.1\times(43.1\times)$ more energy efficient than SMO(C4.5) during training. Overall, combining training and one iteration of inference, Multi-Model HD is $47.1\times$ faster and $47.8\times$ more energy efficient than state-of-the-art multi-label classifiers on average, while providing $5\%$ higher classification accuracy. On the other hand, Power Set HD is $24\times$ faster than state-of-the-art multi-label classifiers on average or approximately $2\times$ slower than Multi-Model HD for $13\%$ higher accuracy.

### C. Multi-label HD in 3D Flash

Figure 5 also shows the latency and energy consumption of Multi-label HD when accelerated in flash. We implement Multi-label HD encoding and training in flash over the three datasets. We observe that our 3D-flash implementation of Multi-label HD is on average $610\times$ faster and $228\times$ more energy-efficient than CPU. Our evaluations show that the performance and energy consumption of Multi-label HD in 3D-flash increases linearly with an increase in the number of training samples. This happens because more data samples result in more huge hypervectors to generate and process. In conventional systems, this translates to a huge amount of data transfers between the core and memory. In contrast, our 3D-flash implementation generates hypervectors (encoding) while reading data out of the slow flash arrays and processes (training) them on the disk itself, reducing data movement.

## VI. Conclusion

In this paper, we design the first accelerator for multi-label HD classification in 3D storage. We also propose two different transformation methods to map HD single label classification to multi-label classification: Power Set HD and Multi-Model HD. Overall, combining training and one iteration of inference, Multi-Model HD is $47.1\times$ faster and $47.8\times$ more energy efficient than state-of-the-art multi-label classifiers, while also achieving $5\%$ higher accuracy on average. Power Set HD can achieve $13\%$ higher accuracy than Multi-Model HD, but is $2\times$ slower. We additionally propose in-3D-flash acceleration that further improves the energy efficiency of Muilti-Model HD training by $228\times$ and speedup by $610\times$.


## Acknowledgements

This work was supported by NSF grants #1527034, #1730158, #1826967, and #1911095.


## References


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

[2] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 3, no. 3, pp. 1–13, 2007.

[3] T. Durand, N. Mehrasa, and G. Mori, "Learning a deep convnet for multi-label classification with partial labels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 647–657, 2019.

[4] Y. Sun *et al.*, "Internet of things and big data analytics for smart and connected communities," *IEEE Access*, vol. 4, pp. 766–773, 2016.

[5] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.

[6] 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.

[7] 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.

[8] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *ICRC*, pp. 1–6, IEEE, 2017.

[9] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, pp. 64–69, ACM, 2016.

[10] I. Jo, D.-H. Bae, A. S. Yoon, J.-U. Kang, S. Cho, D. D. Lee, and J. Jeong, "Yoursql: a high-performance database system leveraging in-storage computing," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 924–935, 2016.

[11] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, *et al.*, "Biscuit: A framework for near-data processing of big data workloads," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 153–165, 2016.

[12] G. Koo, K. K. Matam, I. Te, H. K. G. Narra, J. Li, H.-W. Tseng, S. Swanson, and M. Annavaram, "Summarizer: trading communication with computing near storage," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 219–231, IEEE, 2017.

[13] Z. Ruan, T. He, and J. Cong, "Insider: designing in-storage computing system for emerging high-performance drive," in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, pp. 379–394, 2019.

[14] S. Gupta, J. Morris, M. Imani, R. Ramkumar, J. Yu, A. Tiwari, B. Aksanli, and T. Rosing, "Thrifty: Training with hyperdimensional computing across flash hierarchy," in *Proceedings of the IEEE/ACM 2020 International Conference on Computer-Aided Design (ICCAD)*, 2020.

[15] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *IoT*, p. 38, ACM, 2018.

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

[17] M. Imani *et al.*, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *IEEE BHI*, pp. 271–274, IEEE, 2018.

[18] A. Rahimi *et al.*, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *ICRC*, pp. 1–8, IEEE, 2016.

[19] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 4, pp. 1–25, 2019.

[20] V. S. Mailthody, Z. Qureshi, W. Liang, Z. Feng, S. G. De Gonzalo, Y. Li, H. Franke, J. Xiong, J. Huang, and W.-m. Hwu, "Deepstore: In-storage acceleration for intelligent queries," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 224–238, 2019.

[21] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[22] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *Journal of Machine Learning Research*, vol. 12, pp. 2411–2414, 2011.

[23] S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas, "Protein classification with multiple algorithms," in *Panhellenic Conference on Informatics*, pp. 448–456, Springer, 2005.

[24] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern recognition*, vol. 37, no. 9, pp. 1757–1771, 2004.

[25] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," in *Advances in neural information processing systems*, pp. 681–687, 2002.