StocHD: Stochastic Hyperdimensional System for Efficient and Robust Learning from Raw Data

Prathyush Poduval*, Zhuowen Zou$^{\psi}$, Hassan Najafi*, Houman Homayoun†, Mohsen Imani‡
*Indian Institute of Science, $^{\psi}$UC San Diego, *University of Louisiana, †UC Davis, ‡UC Irvine
Email: m.imani@uci.edu

***Abstract*—Hyperdimensional Computing (HDC) is a neurally-inspired computation model working based on the observation that the human brain operates on high-dimensional representations of data, called *hypervector*. Although HDC is significantly powerful in reasoning and association of the abstract information, it is weak on features extraction from complex data such as image/video. As a result, most existing HDC solutions rely on expensive pre-processing algorithms for feature extraction. In this paper, we propose StocHD, a novel end-to-end hyperdimensional system that supports accurate, efficient, and robust learning over raw data. Unlike prior work that used HDC for learning tasks, StocHD expands HDC functionality to the computing area by mathematically defining stochastic arithmetic over HDC hypervectors. StocHD enables an entire learning application (including feature extractor) to process using HDC data representation, enabling uniform, efficient, robust, and highly parallel computation. We also propose a novel fully digital and scalable Processing In-Memory (PIM) architecture that exploits the HDC memory-centric nature to support extensively parallel computation. Our evaluation over a wide range of classification tasks shows that StocHD provides, on average, 3.3× and 6.4× (52.3× and 143.5×) faster and higher energy efficiency as compared to state-of-the-art HDC algorithm running on PIM (NVIDIA GPU), while providing 16× higher computational robustness.**

I. Introduction

We face increasing needs for efficient processing for diverse cognitive tasks using a vast volume of data generated [1], [2]. However, running machine learning algorithms often results in extremely slow processing speed and high energy consumption on traditional systems or needs a large cluster of application-specific integrated chips (ASIC), e.g., deep learning on Google TPU [3]. There are two key technical challenges that make it difficult to learn in today's computing devices: computation efficiency and robustness to noise.

HyperDimensional Computing (HDC) is introduced as a computational model towards high-efficiency and noise-tolerant computation [4]. HDC is motivated by the observation that the human brain operates on high-dimensional data representations. In HDC, objects are thereby encoded with high-dimensional vectors, called *hypervectors*, which have thousands of elements [5], [6]. It mimics several important functionalities of the human memory model with vector operations, which are computationally tractable and mathematically rigorous in describing human cognition [7], [8].

Although HDC is significantly powerful in reasoning and association of the abstract information, it is weak on features extraction from complex data such as image/video data. This forces HDC algorithms to rely on the pre-processing step to extract useful information from raw data. For an example of image data, HDC solutions rely on signal extracted from popular feature extractor algorithms, such as convolution and histogram of gradient (HOG) [9]. These pre-processing algorithms are not compatible with the HDC learning model, thus need to be processed over traditional data representation. The existing HDC primitives are abstract and approximate, thus cannot support the high precision arithmetic.

Running pre-processing algorithms o traditional data representation provides the following challenges: (i) significant computation cost that dominates the entire learning efficiency, (ii) non-uniform data processing as feature extractors require different hardware optimizations, precision, and data representation, (iii) a low computational robustness coming from non-holographic data representation, making the entire system vulnerable to error, and (iv) the necessity of using expensive data encoding to map extracted features into high-dimensional space. To address these issues, we propose StocHD, a novel end-to-end hyperdimensional learning system operating accurate, efficient, and robust learning over raw generated data. The main contributions are listed below:

- This is the first effort that fundamentally defines stochastic arithmetic over hyperdimensional vectors, enabling highly accurate, efficient, and robust computation. unlike all prior methods that rely on the expensive pre-processing step, StocHD enables an entire learning application (including feature extractor) to process using HDC data representation, enabling uniform, efficient, robust, and parallel computation.
- Our solution mathematically defines stochastic arithmetic over HDC vectors, including weighted addition, subtraction, multiplication, division, and comparisons. We exploit these arithmetic to revisit the pre-processing algorithms to run using uniform HDC data representation without paying the cost of decoding data back to the original space.
- We propose a novel processing in-memory architecture that exploits inherent parallelism and the memory-centric nature of HDC. Our PIM architecture exploits the switching characteristics of non-volatile memory to support tensor-based computation over hypervector internally in memory.

We evaluate StocHD efficiency over a wide range of learning algorithms. Our evaluation shows that StocHD provides, on average, 3.3× and 6.4× (52.3× and 143.5×) faster and higher energy efficiency as compared to state-of-the-art HDC algorithm running on PIM (NVIDIA GTX 1080 GPU). In addition, as compared to state-of-the-art HDC solutions, StocHD provides 16× higher robustness to possible noise.

II. Background and Related Work

Hyperdimensional computing (HDC): is a computational model developed based on the observation that the human brain operates on high-dimensional data [4]. The fundamental units of computation in HDC are "hypervectors", which are constructed using an encoding procedure [6]. Although HDC is significantly powerful in reasoning and association of the

978-1-6654-3274-0/21/$31.00 ©2021 IEEE

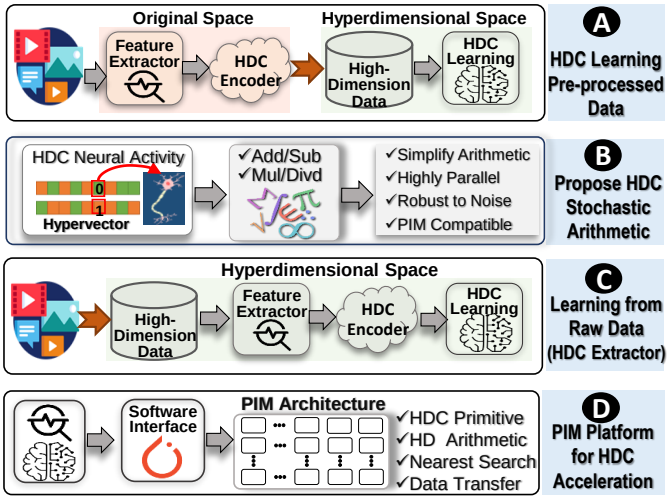

Fig. 1: StocHD framework, an end-to-end HDC learning from raw data along with processing in-memory acceleration

abstract information, it is weak in features extraction from complex data, e.g., image/video. As a result, most existing HDC solutions are operating over costly pre-processed data [5], [10]–[12]. This pre-processing often takes a large portion of the total learning cost.

Stochastic computing (SC): represents numbers in terms of probabilities in long independent bit-streams [13], [14]. SC supports various encoding modules to map binary data to stochastic representation [13]. Arithmetic operations in this representation involve simple logic operations on uncorrelated and independently generated input bit-streams. SC uses a binary sequence to represent real numbers between 0 and 1 via the proportion of 1 bits in the sequence. Arithmetic operations are done using bitwise operations [15], [16].

Although both SC and HDC use similar redundant high-dimensional representation, their goals and strengths are complimentary. *(i) Representation* The SC defines operations over vectors that their percentage of 1s determines their value. While HDC works based on a pattern of neural actively in high-dimensional space, where vectors have roughly equal numbers of 0s and 1s. *(ii) Application:* SC is capable of performing efficient and highly parallel arithmetic operations, enable us to accelerate feature extractor and signal processing applications. In contrast, HDC is a highly approximate cognitive and learning model. This paper is the first effort to fundamentally define SC operations over HDC space, expanding HDC functionality to the computing domain. Our solution enables HDC to process both pre-processing and learning steps, enabling efficient and robust system.

III. StocHD Framework Overview

Figure 1 shows an overview of the StocHD framework consisting of HDC learning and computing solutions. As Figure 1 Ⓐ shows, the existing HDC learning solutions operate over pre-processed data. The pre-processing step is a costly feature extractor performing over original data. HDC implements efficient and robust learning after maps the extracted features into high-dimensional space. In this paper, we propose a novel solution that expands HDC functionality to the computing area, defining stochastic arithmetic operations over HDC vectors (Ⓑ). Our framework translates all data points to hyperdimensional space, enabling both feature extraction and learning to perform using uniform data representation. For example, for image data convolution or Histogram of Gradient (HOG) are commonly used feature extractors. StocHD exploits HDC arithmetic to revisit the feature extractor algorithm to high-dimensional space. As Figure 1 Ⓒ shows, StocHD framework provides: (1) an end-to-end learning solution that enables fully HDC learning over raw data, (2) high computational robustness, as the entire application (including feature extractor) can benefit from the holographic data representation, and (3) significant efficiency as HDC revisits the complex feature extraction with parallel bitwise operations.

HDC has a memory-centric architecture with primitives hardware friendly operations and extensive parallelism [10]. These features make HDC idea for in-memory acceleration. We propose a novel processing in-memory platform that supports all StocHD operations directly over digital data stored in the memory. This eliminates the data movement issue between memory and computing unit, which dominates the entire StocHD energy. In Section V, we show how our PIM platform can accelerate the entire HDC application, from feature extraction to the learning process (Ⓓ).

IV. StocHD Stochastic Primitives

A. HDC Supported Operations

HDC encoding works based on a set of defined primitives [4]. Our goal is to exploit the same primitives to define SC-based arithmetic operations over HDC vectors. HDC is an algebraic structure; it uses search along with several key operations (and their inverses): *Bundling* (+) that acts as memorization during hypervector addition, *Binding* (*) that associates multiple hypervectors, and *Permutation* (ρ) which preserves the position by performing a single rotational shift. In HDC, the hypervectors are compositional – they enable computation in superposition, unlike standard neural representations. These HDC operations allow us to reason about and search through input data that satisfy prespecified constraints. To support arithmetic operation, StocHD requires the following HDC operations:

HDC Hypervector Generation: We generate a random hypervector with elements ± 1 such that $+1$ appears with probability p. This will allow us to construct HDC representations of arbitrary numbers via a D dimensional vector. In our HDC system, information is stored with components ± 1. We fix a random HDC vector $\vec{\mathcal{V}}_1$ to be a Basis vector. A random HDC vector $\vec{\mathcal{V}}_h$ ($h \in [-1, 1]$) is said to represent the number h if $\delta(\vec{\mathcal{V}}_h, \vec{\mathcal{V}}_1) = h$. This is consistent with our notation for $\vec{\mathcal{V}}_1$ which means that $\vec{\mathcal{V}}_1$ represents the number 1. Note that based on our representation, $\vec{\mathcal{V}}_{-a} = -\vec{\mathcal{V}}_a$.

Probabilistic Merging: Given n numbers $\{a_1, a_2, .., a_n\}$ and their corresponding probability values, $\{p_1, p_2, ..., p_{n-1}\} \in [0, 1]$, where p_n is defined by $\sum_{i=1}^{n} p_i = 1$, the probabilistic merging chooses the number a_i with probability p_i. This operation can be extended to operate n hypervectors, where we select each dimension of merged hypervector by probabilistic merging of n elements located in the same dimension of given input hypervectors.

Similarity Measurement: Between two HD vectors $\vec{\mathcal{V}}_1$ and $\vec{\mathcal{V}}_2$, the similarity defines as $\delta(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2) = \frac{\vec{\mathcal{V}}_1 . \vec{\mathcal{V}}_2}{D}$ where D is

the number of dimensions and $(\cdot)$ is the vector dot product operator. HDC supports other similarity metrics, such as Hamming similarity, that measures the number of dimensions at which two HDC vectors differ.

B. HDC Arithmetic Operations

Weighted Average: StocHD defines weighted accumulation over hypervectors. Given two random HDC vectors $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$ and two probability numbers $\{p,\ q\} \in [0,1]$ $(p+q=1)$, we define $C = p\vec{\mathcal{V}}_a \oplus q\vec{\mathcal{V}}_b$ to be the random HDC vector whose i^{th} component is $\vec{\mathcal{V}}_a^i$ or $\vec{\mathcal{V}}_b^i$ with probability p and q, respectively. This can be extended to probabilistic merging of n HDC vectors $\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2, ..., \vec{\mathcal{V}}_n$ and $n-1$ probabilities $p_1, p_2, ..., p_{n-1}$ (p_n is defined by $\sum_{i=1}^{n} p_i = 1$). We can similarly define the weighted sum as $p_1\vec{\mathcal{V}}_1 \oplus p_2\vec{\mathcal{V}}_2 \oplus \cdots \oplus p_n\vec{\mathcal{V}}_n$.

Let us consider $\vec{\mathcal{V}}_c = 0.5\vec{\mathcal{V}}_a \oplus 0.5\vec{\mathcal{V}}_b$. To verify the correct functionality of StocHD, we need to show that $c = \frac{a+b}{2}$. Based on our definition, $\vec{\mathcal{V}}_a$ has similarity a with $\vec{\mathcal{V}}_1$. As a result it has exactly $\frac{a+1}{2}$ parts components common with $\vec{\mathcal{V}}_1$. So if we randomly chose a dimension i, the probability that the i^{th} component of both $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_1$ match is given by $\frac{1+a}{2}$. Considering the i^{th} component of $\vec{\mathcal{V}}_c$, the probability that the i^{th} component is taken from $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$ is 0.5 and 0.5 respectively. Thus, the probability that the i^{th} component of $\vec{\mathcal{V}}_c$ matches with $\vec{\mathcal{V}}_1$ is given by $\frac{1}{2}\frac{1+a}{2} + \frac{1}{2}\frac{1+b}{2} = \frac{1+\frac{a+b}{2}}{2}$. As a result, $\delta(\vec{\mathcal{V}}_c, \vec{\mathcal{V}}_1) = \frac{a+b}{2}$ which is what was claimed. Similarly, StocHD supports weighted subtraction given by $0.5v_a \oplus 0.5v_{-b} = v_{\frac{a-b}{2}}$.

Constructing Representations: Let us define $\vec{\mathcal{V}}_a = \frac{a+1}{2}\vec{\mathcal{V}}_1 \oplus \frac{1-a}{2}(-\vec{\mathcal{V}}_1)$. Note that $\vec{\mathcal{V}}_a$ will have $\frac{1+a}{2}$ components same with $\vec{\mathcal{V}}_1$ and $\frac{1-a}{2}$ component same with $-\vec{\mathcal{V}}_1$ (which has components complementary to $\vec{\mathcal{V}}_1$). As a result we have $\delta(\vec{\mathcal{V}}_a, \vec{\mathcal{V}}_1) = a$ and thus we have constructed a representation of the number a given by $\vec{\mathcal{V}}_a = \frac{a+1}{2}\vec{\mathcal{V}}_1 \oplus \frac{1-a}{2}(\vec{\mathcal{V}}_{-1})$. Note that if $a \in [0,1]$, then this is equivalent to $\frac{1\pm a}{2} \in [0,1]$ and so the probabilities for merging are well defined. This operation will be the building blocks of all other arithmetic operations.

Multiplication: Given two HDC representations $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$, we show a way to construct $\vec{\mathcal{V}}_{ab}$. Consider the i^{th} dimension of $\vec{\mathcal{V}}_c$ and set it to the i^{th} dimension of $\vec{\mathcal{V}}_1$ if the i^{th} dimension of $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$ are both $+1$ or both -1. Otherwise set the i^{th} dimension of $\vec{\mathcal{V}}_c$ to be the i^{th} dimension of $\vec{\mathcal{V}}_{-1}$. From this construction, the probability that the i^{th} dimension of $\vec{\mathcal{V}}_c$ is the same as $\vec{\mathcal{V}}_1$ is given by $\frac{ab+1}{2}$. Thus, $\delta(\vec{\mathcal{V}}_c, \vec{\mathcal{V}}_1) = 2\frac{ab+1}{2} - 1 = ab$ and so $\vec{\mathcal{V}}_c \equiv \vec{\mathcal{V}}_{ab}$. In a simpler form, $\vec{\mathcal{V}}_{ab}$ can be computed by element-wise product of $\vec{\mathcal{V}}_a$, $\vec{\mathcal{V}}_b$, and $\vec{\mathcal{V}}_1$ hypervectors.

Comparisons: Comparison is another key operation for data processing as well as required operation to implement division. Suppose we are given hypervectors $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$ corresponding to a and b values in original space. We can check comparison by first calculating $\vec{\mathcal{V}}_{\frac{a-b}{2}} = \vec{\mathcal{V}}_a \oplus \vec{\mathcal{V}}_{-b}$. Then we evaluate the value using $\frac{a-b}{2} \sim \delta(\vec{\mathcal{V}}_{\frac{a-b}{2}}, \vec{\mathcal{V}}_1)$. Finally, we can check whether $\frac{a-b}{2}$ is positive, negative or 0.

Division: Consider two vectors $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$. Our aim is to construct an HDC vector $\vec{\mathcal{V}}_{a/b}$, which would be saturated if a/b lies outside the range of our HDC arithmetic system. Without loss of generality, assume a and b are both positive. Then, we do the following steps:

- Initialize two vectors $\vec{\mathcal{V}}_{low}$ to be $\vec{\mathcal{V}}_{-1}$ and $\vec{\mathcal{V}}_{high}$ to be $\vec{\mathcal{V}}_1$
- Calculate $\vec{\mathcal{V}}_{mid} = 0.5\vec{\mathcal{V}}_{low} \oplus 0.5\vec{\mathcal{V}}_{high}$ and $\vec{\mathcal{V}}_{midb} = \vec{\mathcal{V}}_{mid} \otimes \vec{\mathcal{V}}_b$ where we use $\otimes$ to stand for multiplication
- If $\vec{\mathcal{V}}_{midb} > \vec{\mathcal{V}}_a$, then do $\vec{\mathcal{V}}_{high} \to \vec{\mathcal{V}}_{midb}$
- If $\vec{\mathcal{V}}_{midb} < \vec{\mathcal{V}}_a$, then do $\vec{\mathcal{V}}_{low} \to \vec{\mathcal{V}}_{midb}$
- Repeat from step 2 until $\vec{\mathcal{V}}_{midb} = \vec{\mathcal{V}}_a$ or $\vec{\mathcal{V}}_c = \vec{\mathcal{V}}_{\pm 1}$, then stop.

This process eventually has to end because the difference between the evaluation of $\vec{\mathcal{V}}_{low}$ and $\vec{\mathcal{V}}_{high}$ keeps decreasing at each iteration. Thus, we get a representation of a/b which is $\vec{\mathcal{V}}_{mid}$.

Doubling: Similar to how we defined division, we can also define a doubling formula to construct $\vec{\mathcal{V}}_{2a}$ from $\vec{\mathcal{V}}_a$. The way to proceed would be to compare $\vec{\mathcal{V}}_{mid/2}$ with $\vec{\mathcal{V}}_a$ rather than $\vec{\mathcal{V}}_{midb}$ in the algorithm for division.

C. Arithmetic Error Rates

We first discuss the error rates for generating errors. We generate a HD-SC representation of a number $a \in [-1,1]$ using $\vec{\mathcal{V}}_a = \frac{a+1}{2}\vec{\mathcal{V}}_1 \oplus \frac{1-a}{2}(\vec{\mathcal{V}}_{-1})$. Let X_i be a random variable with value 1 if the i^{th} dimension of $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_1$ are the same, and -1 otherwise. Moreover, let $S = \frac{\sum_i X_i}{D}$. We note that $\delta(\vec{\mathcal{V}}_a, \vec{\mathcal{V}}_1) = 2S-1$. Now, X_i are independently and identically distributed Bernoulli random variable with $p = \frac{1+a}{2}$, $\mu = \frac{1+a}{2}, \sigma = \frac{\sqrt{1-a^2}}{2}$. Using the Central Limit Theorem, we get $N(0,1)$ is normal distributed. As a result, we have:

$$\mathbb{P}\left(|\delta(\vec{\mathcal{V}}_a, \vec{\mathcal{V}}_1) - a| \geq \frac{2\sigma\epsilon}{\sqrt{D}}\right) = \frac{1}{\sqrt{2\pi}}\int_{\epsilon}^{\infty} e^{-\frac{x^2}{2}}dx$$

The similarity of two vectors $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$ is calculated using $\delta(\vec{\mathcal{V}}_a, \vec{\mathcal{V}}_b) = \sum_{i=1}^{D} \frac{\vec{\mathcal{V}}_a^i \cdot \vec{\mathcal{V}}_b^i}{D}$, Where $\vec{\mathcal{V}}_a^i$ is the i^{th} component of the vector $\vec{\mathcal{V}}_a$. We calculate the Mean Absolute Error (MRE) of the representation $\vec{\mathcal{V}}_a$ representing the number $a \in [-1,1]$ using the formula

$$\frac{\mathbb{E}(|\delta(\vec{\mathcal{V}}_a, \vec{\mathcal{V}}_1) - a|)}{2} \times 100 = \sum_{i=1}^{N} \frac{|\delta(\vec{\mathcal{V}}_{a_i}, \vec{\mathcal{V}}_1) - a_i|}{2N} \times 100$$

Here, we divide by 2 so that we normalise the length of the interval to 1. We use this metric to compare our errors with other methods.

Addition/Multiplication: The error in weighted addition follows the same theoretical analysis of the generational error. This is because the analysis only depends on the relation between the probability with which $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_1$ have a common

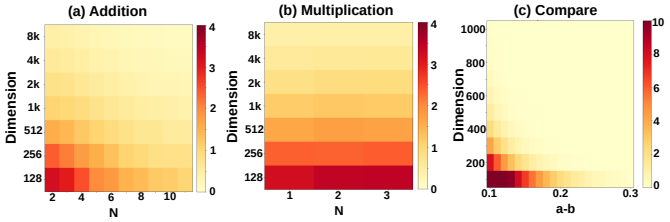

Fig. 2: Error for (a) N-Addition, (b) N-Multiplication, and (c) comparison between two numbers.

dimension, and the value of a itself. The additional advantage is that the repeated weighted addition does not result in an accumulation of error, which is essential in multiplication where we use weighted addition multiple times in a sequence. This arises theoretically from the fact that the distribution of the components of the added vectors follows the correct Bernoulli distribution; thus, the same explained error analysis still holds. Figure 2a shows our error analysis for the errors of the weighted addition of N numbers as a function of the dimension. Our results indicate that the larger the N, the lower the error becomes.

Comparision/Division: Our goal is to find the probability that the comparison returns the correct result. We also recall that $\delta(\vec{\mathcal{V}}_{\frac{a-b}{2}}, \vec{\mathcal{V}}_1)$ is normally distributed with $\mu = \frac{a-b}{2}$ and standard deviation $\sigma = \sqrt{1-(\frac{a-b}{2})^2}/\sqrt{D}$. We assume $\sqrt{1-(\frac{a-b}{2})^2} \sim 1$ which will give us the upper bound for the error. The first case is when $\frac{a-b}{2}$ is positive. The probability that the comparison returns the incorrect value is given by

$$\mathbb{P}(\delta(\vec{\mathcal{V}}_{\frac{a-b}{2}}, \vec{\mathcal{V}}_1) < 0) = \sqrt{\frac{D}{2\pi}} \int_{-\infty}^{0} e^{-\frac{D}{2}\left(x-\frac{a-b}{2}\right)^2} dx$$

The second case is when $\frac{a-b}{2}$ is negative. The probability that the comparison returns the incorrect value can be computed in a similar way. Figure 2c shows the error rate of StocHD comparison as a function of closeness of a and b, and the hypervector dimensions. Note that, although comparison has a much higher error rate in $D = 1024$, division does not see any appreciable loss of error.

V. Processing In-Memory Architecture

In this section, we present a digital-based processing in-memory architecture, called StocHD, which accelerates a wide range of HDC-based algorithms on conventional crossbar memory. StocHD supports all essential HDC operations in memory in a parallel and scalable way. Figure 1 demonstrates an overview of the proposed learning system.

A. StocHD *Accelerator*

Our digital-based PIM architecture enables parallel computing and learning over the hypervectors stored in memory. Unlike prior PIM designs that use large ADC/DAC blocks for analog computing [17], [18], StocHD performs all HDC computations on the digital data stored in memory. This eliminates ADC/DAC blocks, resulting in high throughput/area and scalability. StocHD supports several fundamental operations required to accelerate HDC. StocHD uses two blocks for performing the computation; a *compute block* and a *search block*. Each block supports the following set of operations (shown in Figure 3a): (i) *Arithmetic operations*: row-parallel NOR-based operation. and (ii) *search-based operations*: row-parallel nearest Hamming distance search.

Row-Parallel PIM-based Arithmetic: StocHD supports arithmetic operations directly on digital data stored in memory without reading them out of sense amplifiers [19], [20]. Our design exploits the memristor switching characteristic to implement NOR gates in digital memory [19]. StocHD selects two or more columns of the memory as input NOR operands by connecting them to ground voltage (Shown in Figure 3b). During NOR computation, the output memristor is

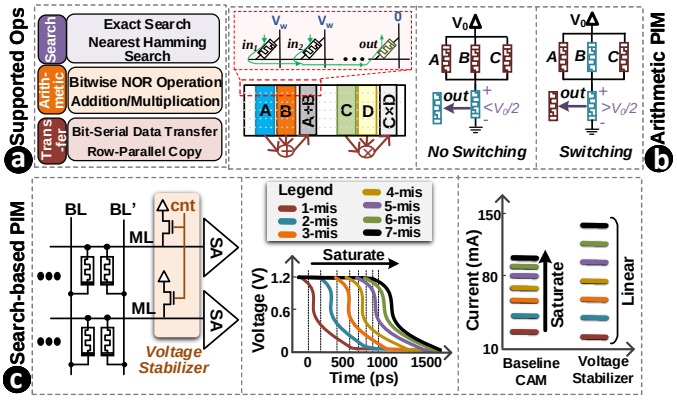

Fig. 3: PIM architecture: (a) supported operations, (b) arithmetic computing, (b) CAM-based search operation, (d) timing characteristic of CAM during search. [21]

switched from R_{ON} to R_{OFF} when one or more inputs stored '1.' value (R_{ON}). In fact, the low resistance input passes a current through an output memristor resulting in writing R_{off} value on it. This NOR computation performs in row-parallel on all the activated memory rows by the row-driver. Since NOR is a universal logic gate, it can be used to implement other logic operations like AND and XOR operations required for HDC automatics. Note that all these arithmetic can be supported in parallel over all dimensional of hypervectors, enabling significant computational speedup.

Nearest Search: The exact search is one of the native operations supported by crossbar memory. During the search, a row-driver of the CAM block pre-charges all CAM rows (match-lines:MLs) to supply voltage (V_{dd}). Consider a CAM cell (shown in Figure 3c), if a query input matches with the stored value in the CAM cell, the ML voltage will stay charged. However, in case of a mismatch between the CAM cell and the query data, the CAM starts discharging the ML. Conventionally, CAM blocks exploit the ML discharging current to enable the exact search operation. Here, we exploit the timing characteristic of each row discharging current to detect a row with minimum distance to query data [21]. As shown in Figure 3c, a CAM row with the minimum discharging current will have the highest similarity with the query data. However, due to reducing the voltage of ML during the search, the ML discharging current saturates with increasing the number of mismatches. This eliminates finding a row with minimum Hamming distance. To provide a more reliable search, we exploit a voltage stabilizer [21] in each CAM row that ensures a fixed ML voltage during the search. We also utilize ganged-circuits [22], as a CAM sense amplifier to enable the nearest search in a row-parallel way.

B. *Architecture*

StocHD exploits row-parallel PIM-based NOR operation to accelerate feature extractors, which are mainly based on arithmetic operation, i.e., bitwise operations in HDC space. The feature extraction can perform by simple bitwise operation between hypervectors representing the values. Next, StocHD supports permutation and row-parallel XOR operation over the high-dimensional features. For example, in case of n extracted features, $\{\vec{f}_1, \vec{f}_2, \cdots, \vec{f}_n\}$, StocHD encodes the information by: $\vec{\mathcal{H}} = \vec{f}_1 \oplus \rho^1 \vec{f}_2 \oplus \cdots \rho^{n-1} \vec{f}_n$, where ρ^n denotes n-

TABLE I: Datasets (n: feature size, k: number of classes)

	n	k	Feature Extractor	Train Size	Description
MNIST	784	10	Convolution	60,000	Handwritten Recognition [26], [27]
UCIHAR	561	12	Noise Filter	6,213	Activity Recognition(Mobile) [28]
ISOLET	617	26	MFCC	6,238	Voice Recognition [29]
FACE	608	2	HOG	522,441	Face Recognition [30]
PAMAP	75	5	FFT	611,142	Activity Recognition(IMU) [31]

bit rotational shift. All encoding steps can perform using row-parallel NOR operation and shift operation that can be implemented by our PIM. StocHD performs classification by checking the similarity of an encoded query with a binary HDC class hypervectors. A query will assign to data with the highest Hamming distance similarity. The inference can be supported using the nearest search supported by our PIM.

VI. Experimental Evaluation

A. Experimental Setup

We implement StocHD using both software and hardware support. In software, we developed a PyTorch-based library of Hyperdimensional computing, supporting all required computing and learning operations. In hardware, we design a cycle-accurate simulator based on PyTorch [23] that emulates StocHD functionality during classification. For hardware, we use HSPICE for circuit-level simulations to measure the energy consumption and performance of all the StocHD operations in 28nm technology. We used system Verilog and Synopsys *Design Compiler* [24] to implement and synthesize the StocHD controller. At the circuit-level, we simulate the cost of inter-tile communication, while in architecture, we model and evaluate intra-tile communications. StocHD works with any bipolar resistive technology, which is the most commonly used in existing NVMs. In order to have the highest similarity to commercially available 3D Xpoint, we adopt the memristor device with a VTEAM model [25].

We evaluate StocHD accuracy and efficiency on five popular datasets such as a large data that includes hundreds of thousands of facial data. Table I lists the workloads, their corresponding feature extractors, and dataset size.

B. StocHD Learning Accuracy

State-of-the-art Learning Algorithms: We compare StocHD classification accuracy with state-of-the-art learning algorithms, including Deep Neural Networks (DNN), Support Vector Machine (SVM), and AdaBoost. The DNN models are trained with Tensorflow, and we exploited the Scikit-learn library to train other ML algorithms. We exploit the grid search to identify the best hyper-parameters for each model. Our evaluation shows that StocHD provides comparable accuracy to other state-of-the-art algorithms (only 0.2% lower than DNN, and 1.5% and 1.8% higher than SVM and AdaBoost).

Baseline HDC Algorithms: we compare HDC classification accuracy in different configurations: (i) without feature extractor where learning directly happens over a raw data, (ii) with feature extractor running on original data, and (iii) using StocHD arithmetic computation to processed feature extraction. Our evaluation shows that HDC with no feature extraction provides, on average, 59% lower accuracy than HDC operating over extracted features. Revisiting the feature extractor with StocHD stochastic arithmetic can almost provide the same result as running feature extraction over original data. The quality of StocHD computation depends on the HDC dimensionality. Using $D = 4,000$ dimensions, StocHD provides the same accuracy as the baseline algorithm. Reducing dimension to $D = 3,000$ and $D = 2,000$ reduces StocHD accuracy, on average, by 0.9% and 2.1%, respectively. This lower accuracy comes from StocHD accumulative noise during the pre-processing step.

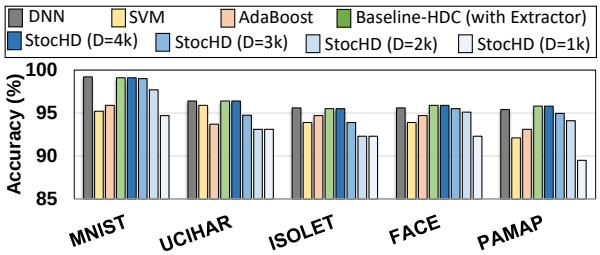

Fig. 4: Comparing StocHD accuracy to state-of-the-art.

C. StocHD Learning Efficiency

PIM & Feature Extraction: Figure 5 compares StocHD efficiency with the baseline HDC running on the proposed PIM platform. All results are normalized to execution time and energy consumption of the baseline HDC running on NVIDIA GTX 1080 GPU. In GPU, the feature extraction takes, on average, 72% of execution time and 77% of total learning energy consumption. As explained in Section V, our proposed PIM is an in-memory computing platform that can accelerate any tensor-based algorithms, including the feature extractors, listed in Table I. To accelerate feature exaction, PIM exploits high-precision arithmetic computation, such as addition and multiplication, that operates over original data representation. However, PIM is sequential and slow in supporting the high-precision arithmetic over traditional data. For example, for N-bit addition and multiplication, PIM requires $13N + 1$ and $13N^2 + 16N + 1$ NOR cycles, respectively. This makes our PIM platform less ideal to operate over traditional data, e.g., fixed point or floating-point representation.

StocHD Feature Extraction: Figure 6 shows the breakdown of the execution time in the baseline HDC and StocHD running on GPU and PIM platform. Our evaluation shows that the slowness of the PIM to support high-precision arithmetic further increases a portion that feature extractor from the total execution time (88% over all tested applications). In contrast, StocHD is an end-to-end HDC-based platform that speeds up the feature extraction by simplifying the arithmetic operation to highly parallel bitwise operations. The HDC arithmetic are extensively parallel and PIM friendly. For example, unlike multiplication in original space that performs bit-sequentially, PIM can implement StocHD stochastic multiplication with two AND operations over hypervectors. Our evaluation shows that StocHD using $D = 4,000$ provides 3.3× and 6.4× (52.3× and 143.5×) faster and higher energy efficiency as compared to baseline HDC running on the same PIM (GPU) platform.

Breakdown: Our evaluation on Figure 6 shows that StocHD not only reduces the cost of feature extractor but also diminishes the cost of the encoding module. In baseline HDC, the encoding requires an extra step for feature quantization and non-linear data mapping. In contrast, in StocHD, features are already in high-dimensional space. Therefore, a linear HDC encoding can aggregate the high-dimensional features extracted by our pre-processing method. Our evaluation shows

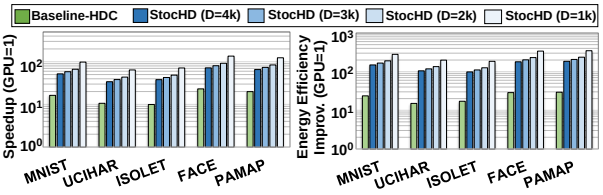

Fig. 5: StocHD efficiency running on PIM platform.

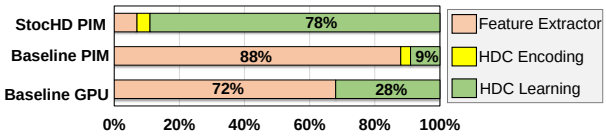

Fig. 6: Breakdown of execution time.

that StocHD reduces the performance overhead of the feature extractor and encoding to less than 7% and 4%, respectively. **Dimensionality:** Reducing the dimensionality improves StocHD computation efficiency. As Figure 5 shows, StocHD using $D = 2,000$ dimensions provides, on average, 4.2× and 8.1× (67.0× and 183.9×) faster and higher energy efficiency as compared to the baseline HDC running on PIM (GPU).

D. StocHD *Robustness*

Many advanced technologies typically pose issues for hardware robustness [32]. One of the main advantages of StocHD is its high robustness to noise and failure in hardware. In StocHD, hypervectors are random and holographic with i.i.d. components. Each hypervector stores all the information across all its components so that no component is more responsible for storing any piece of information than another. This makes a hypervector robust against errors in its components. StocHD efficiency and robustness highly depend on the dimensionality and the precision of each hypervector element. Table II compares StocHD robustness to noise in the memory devices. StocHD provides significantly higher robustness to memory noise than the baseline HDC algorithm. In binary representation, an error only flips a reference dimension results in minor changes in the entire hypervector pattern. In contrast, an error in original space (feature extractor in baseline HDC) can happen in most significant bits, which significantly affects the absolute value and robustness. Our results indicate that 10% failure in memory cells results in 0.9% and 14.4% loss on StocHD and the baseline HDC accuracy.

Table II also explores the impact of limited NVM endurance on StocHD quality of learning. We assume an endurance model with $\mu = 10^7$ [33]. Our evaluation shows that after a few years of using our PIM-based platform, similar to the human brain, StocHD starts forgetting information stored in reference hypervector. To address this issue, we perform wear-leveling to distribute writes uniformly over memory blocks. The overhead of wear-leveling is minor as (i) StocHD has predictable write pattern, and (ii) wear-leveling can happen in long-time periods. Our evaluation shows that the baseline HDC has higher sensitivity to the endurance issue. This is because feature extractor requires PIM arithmetic operation that involves several device switching. In contrast, StocHD computes feature extraction with minimal write operation.

VII. Conclusion

We propose StocHD, a novel end-to-end hyperdimensional system that supports accurate, efficient, and robust learning over raw data. StocHD expands HDC functionality to the computing area by mathematically defining stochastic arithmetic operations over HDC hypervectors. StocHD enables an entire learning application (including feature extractor) to process using HDC data representation, enabling uniform, efficient, robust, and highly parallel computation.

TABLE II: Quality loss using noisy and low endurance.

Memory Error	**1%**	**2%**	**5%**	**10%**	**15%**
Baseline HDC	1.1%	4.5%	9.3%	14.4%	27.3%
StocHD ($D = 4k$)	0.0%	0.0%	0.3%	0.9%	2.1%
StocHD ($D = 1k$)	0.0%	0.2%	0.8%	1.8%	3.4%
Endurance Years	**1**	**2**	**3**	**4**	**5**
Baseline HDC	0%	1.7%	4.1%	11.5%	24.5%
StocHD ($D = 4k$)	0%	0.5%	0.9%	1.8%	2.4%
StocHD ($D = 0.5k$)	0.0%	0.8%	1.6%	3.3%	5.2%

Acknowledgment

This work was partially supported by Semiconductor Research Corporation Task No. 2988.001, Department of the Navy, Office of Naval Research, grant #N00014-21-1-2225, National Science Foundation grant 2019511, and Louisiana Board of Regents Support Fund LEQSF(2020-23)-RD-A-26.

References

[1] X.-W. Chen and X. Lin, "Big data deep learning: challenges and perspectives," *IEEE access*, vol. 2, pp. 514–525, 2014.
[2] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.
[3] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*. IEEE, 2017, pp. 1–12.
[4] 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.
[5] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *CLOUD*. IEEE, 2019, pp. 435–446.
[6] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*. ACM, 2016, pp. 64–69.
[7] A. Mitrokhin *et al.*, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, vol. 4, no. 30, 2019.
[8] M. Imani *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *HPCA*. IEEE, 2021.
[9] S. Hou *et al.*, "Dualnet: Learn complementary features for image recognition," in *ICCV*, 2017, pp. 502–510.
[10] H. Li *et al.*, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *IEDM*. IEEE, 2016, pp. 16–1.
[11] M. Nazemi *et al.*, "Synergiclearning: Neural network-based feature extraction for highly-accurate hyperdimensional learning," *arXiv preprint arXiv:2007.15222*, 2020.
[12] A. Hernandez-Cano *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *DATE*. IEEE, 2021.
[13] A. Alaghi *et al.*, "Survey of stochastic computing," *ACM TECS*, vol. 12, no. 2s, pp. 1–19, 2013.
[14] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *TCAS I*, vol. 37, no. 8, pp. 1515–1531, 2017.
[15] K. Kim *et al.*, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *DAC*, 2016, pp. 1–6.
[16] S. Liu *et al.*, "Energy efficient stochastic computing with sobol sequences," in *DATE*. IEEE, 2017, pp. 650–653.
[17] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*. IEEE, 2016, pp. 14–26.
[18] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*. IEEE Press, 2016, pp. 27–39.
[19] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*. ACM, 2019, pp. 802–815.
[20] S. Kvatinsky *et al.*, "Magic—memristor-aided logic," *TCAS II*, vol. 61, no. 11, pp. 895–899, 2014.
[21] M. Imani *et al.*, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *MICRO*. IEEE, 2020, pp. 356–371.
[22] M. Imani, X. Yin *et al.*, "Searchd: A memory-centric hyperdimensional computing with stochastic training," *TCAD*, vol. 39, no. 10, pp. 2422–2433, 2019.
[23] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NIPS*, 2019, pp. 8026–8037.
[24] "Synopsys," http://www. synopsys. com.
[25] S. Kvatinsky *et al.*, "Vteam: A general model for voltage-controlled memristors," *TCAS II*, vol. 62, no. 8, pp. 786–790, 2015.
[26] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[27] D. Ciregan *et al.*, "Multi-column deep neural networks for image classification," in *CVPR*. IEEE, 2012, pp. 3642–3649.
[28] D. Anguita *et al.*, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *AAL*. Springer, 2012, pp. 216–223.
[29] "Uci machine learning repository," http://archive.ics.uci.edu/ml/datasets/ISOLET.
[30] A. Angelova *et al.*, "Pruning training sets for learning of object categories," in *CVPR*. IEEE, 2005.
[31] A. Reiss *et al.*, "Introducing a new benchmarked dataset for activity monitoring," in *ISWC*. IEEE, 2012, pp. 108–109.
[32] T. F. Wu *et al.*, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *ISSCC*. IEEE, 2018, pp. 492–494.
[33] J. B. Kotra *et al.*, "Re-nuca: A practical nuca architecture for reram based last-level caches," in *IPDPS*. IEEE, 2016, pp. 576–585.