ADVANCED INTELLIGENT SYSTEMS
Open Access
www.advintellsyst.com

NetHD: Neurally Inspired Integration of Communication and Learning in Hyperspace

Prathyush P. Poduval, Yang Ni, Zhuowen Zou, Kai Ni, and Mohsen Imani*

The 6G network, the next-generation communication system, is envisaged to provide unprecedented experience through hyperconnectivity involving everything. The communication should hold artificial intelligence-centric network infrastructures as interconnecting a swarm of machines. However, existing network systems use orthogonal modulation and costly error correction code; they are very sensitive to noise and rely on many processing layers. These schemes impose significant overhead on low-power internet of things devices connected to noisy networks. Herein, a hyperdimensional network-based system, called *NetHD*, is proposed, which enables robust and efficient data communication/learning. *NetHD* exploits a redundant and holographic representation of hyperdimensional computing (HDC) to design highly robust data modulation, enabling two functionalities on transmitted data: 1) an iterative decoding method that translates the vector back to the original data without error correction mechanisms, or 2) a native hyperdimensional learning technique on transmitted data with no need for costly data decoding. A hardware accelerator that supports both data decoding and hyperdimensional learning using a unified accelerator is also developed. The evaluation shows that *NetHD* provides a bit error rate comparable to that of state-of-the-art modulation schemes while achieving 9.4 × faster and 27.8 × higher energy efficiency compared to state-of-the-art deep learning systems.

1. Introduction

The 6G network, the next-generation communication system, is envisaged to provide unprecedented experience through hyperconnectivity involving everything. Industries and academies have envisioned that next-generation networks will shift the paradigm in conventional wireless communications.[1–3] It does not mean just faster communication with the developments of high-performance communication links on terahertz bands and novel antenna technology.[4] Still, it should hold for artificial intelligence (AI)-centric network infrastructures as interconnecting a swarm of machines that are growing exponentially to collect massive data from environments.[5–9] This trend essentially poses the following challenges: 1) Low latency communication: To serve advanced applications such as on-device AI, extended reality, and real-time multimedia, 6G should provide an ultralow user-experienced latency of less than 10 ms while competing with a higher data rate of around 1000 Gbps. However, state-of-the-art network protocols are not ideal for achieving such latency due to complex data modulation/demodulation that involves costly iterative procedures with error correction codes.[10–13] 2) Reliable networking: Because of the tremendous volume of data transferred, network interference and noise are unavoidable and are rather significantly intensified to realize future duplex technology.[14–16] 3) Learning integration: Today's communication systems rely on many layers of information processing, from data compression and modulation by senders through demodulation and decompression by receivers to data processing, e.g., machine learning (ML). Unfortunately, the communication and learning processes are kept and optimized separately in existing systems. Therefore, relying only on the improvement of mobile devices' battery life, which is not fast enough given the rapid AI evolution, does not suffice to meet their extensive demand for efficient processing.

Hyperdimensional modulation (HDM) is introduced as a new modulation scheme designed for ultrareliable low-latency communication.[13,17,18] HDM already showed more reliability than binary phase-shift keying (BPSK), protected by state-of-the-art low density parity check (LDPC) and polar error correction codes for the same spectral efficiency.[13] In addition, HDM has a lower complexity than LDPC, Polar, and convolutional codes. However, there is a crucial challenge in the existing HDM modulations when applying them to the IoT applications that involve both communication and data assimilation on less powerful devices—the HDM decoding or demodulation is a costly iterative process that involves an extensive search for noise cancelation.

P. P. Poduval, Y. Ni, Z. Zou, M. Imani
Department of Computer Science
University of California Irvine
Irvine, CA 92697, USA
E-mail: ppoduval@uci.edu

K. Ni
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556, USA

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.202300841.

DOI: 10.1002/aisy.202300841

It implies that end-to-end tasks should pay the enormous computation cost for decoding data even before starting the learning process, which generates a higher computational complexity.

In this work, we focus on advanced technology for communication, which integrates data modulation with ML on the same horizon toward hyper-reliable communication and efficient AI processing. Our solution is based on hyperdimensional computing (HDC), which is an alternative computing paradigm inspired by neurological human memory models.[19] Many academic and industry works have examined HDC to utilize high-dimensional data representation, which is known to be the way the human brain operates.[20–28] They showed that HDC offers high robustness in data processing and effectively realizes ML by imitating human cognition with mathematically rigorous vector operations that describe human memory capabilities to store, load, and compare various information. HDC is well suited to address communication and learning challenges in networking systems, as 1) HDC representation is holographic in that it spreads information over high-dimensional components, thus providing strong robustness to noise[27,29–32]—a key strength to realize reliable communication, 2) it offers an intuitive and human-interpretable learning process[23] which is computationally efficient to train and highly parallel at heart,[33–36] and 3) HDC can naturally enable lightweight privacy and security.[21,37]

Utilizing the advantages of HDC, we propose *NetHD*, a HDC-based network system for robust and efficient data communication and learning. *NetHD* develops novel encoding methods that map data into high-dimensional space and transmit the encoded data through the network. The transmitted data can be accurately decoded back to the original space at the destination node, or more importantly, it can be directly used to perform learning tasks. The main contributions of the article are listed below:

We design a novel encoding method that exploits redundant and holographic HDC representation for ultraefficient and robust data communication. Our encoder utilizes a symbolic HDC representation to distribute information among long vectors. We also propose a decoding method that recovers originally transmitted data. As HDC encoding spreads the data over a large hypervector, we can preserve sufficient information even when a substantial number of bits can be corrupted, resulting in high noise robustness for low signal–noise ratio (SNR) scenarios.

Unlike existing learning solutions that aim to optimize computation and communication separately, we introduce a novel approach that fundamentally merges data modulation and learning. *NetHD* implements hyperdimensional learning directly on transmitted data without costly iterative data decoding. In addition, *NetHD* exploits the robustness of hyperdimensional learning to enable holographic and highly compressed data communication. We show how *NetHD* can enable classification/clustering over compressed transmitted data, thus significantly improving total system efficiency.

To enable fast and efficient data decoding/learning, we design a hardware accelerator based on ferroelectric memory devices. It offloads search-based operations of our decoding mechanism to content addressable memory (CAM), supporting row parallel search operations. Our CAM supports the nearest search over complex-valued hypervector variables, resulting in high efficiency.

We evaluated *NetHD* over a wide range of network conditions and under various SNR scenarios. Our evaluation shows that *NetHD* provides a bit error rate comparable to that of the state-of-the-art modulation schemes while fundamentally merging HDM and learning. Our evaluation shows that *NetHD* achieves 9.4× and 27.8× faster and higher energy efficiency compared to deep neural network (DNN), respectively. Our proposed CAM-based hardware accelerator results in 108.3× and 27.1× (35.8× and 22.0×) faster and higher energy efficiency during data decoding (learning) than embedded graphical processing unit (GPU).

2. NetHD Design

2.1. Overview

Figure 1a shows an overview of the existing systems using non-integrated communication and ML. The processing pipeline at the transmitter starts with data compression and data encoding. The channel can often be an interference channel that adds both

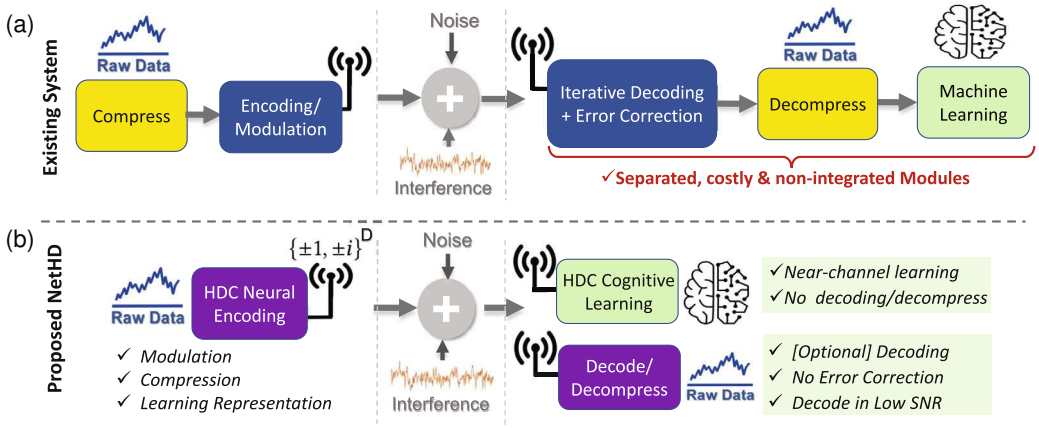

Figure 1. a) An overview of existing systems using a deep communication-learning pipeline. b) Our proposed *NetHD* integrates communication and ML using HDC.

noise and interference to the encoded signal. In the receiver, the system first decodes the received signal using costly iterative decoding coupled with an error correction mechanism. The decoded data will then be decompressed back before being processed by ML. Unfortunately, existing communication protocols use layered processing for data communication isolated from ML.

Envisioning that future network should involve both communication and learning infrastructures, we will exploit rigorous mathematics which describes data representation during the communication and computation during learning procedures based on high-dimensional vectors. We propose a hyperdimensional network-based system, called *NetHD*, that enables robust and efficient data communication/learning. Unlike existing communication protocols that use layered processing for data communication, *NetHD* fundamentally integrates communication with ML. Figure 1b shows an overview of the *NetHD* framework. The first step of *NetHD* is to encode the data in a redundant high-dimensional representation. This high-dimensional representation stores information in a holographic manner, preserving sufficient information even when a substantial number of hypervector elements are corrupted. *NetHD* uses the encoded data to transfer them through the network. The network is often a noisy channel that adds both noise and interference to the encoded signal. The receiver gets noisy data and has two options. First, we can decode the data back to the original space. We propose a lightweight iterative method that decodes the transmitted data without using any error correction. Our decoding solution is significantly robust against low SNR networks and interference. Second, we can also directly operate hyperdimensional learning over the encoded data without the need for costly data decoding. *NetHD* enables various hyperdimensional learning over transmitted data, including classification and clustering. We also introduce the idea of dynamic data compression in *NetHD* encoding to trade accuracy and communication cost.

2.2. Preliminary of HDC

Brain-inspired HDC uses distributed high-dimensional representations of data called "hypervectors" as its fundamental units of computation, based on the observation that the human brain operates in a similar way.[19] These hypervectors are constructed using an encoding procedure, and there are many nearly orthogonal options with thousands of dimensions, allowing the representation of target data with different points in hyperspace.[38–40]

2.2.1. Hyperdimensional Arithmetic

The HDC learning utilizes well-defined vector space operations to combine hypervectors into a new hypervector while keeping the information with high probability. Assume that $\vec{\mathcal{H}}_1$ and $\vec{\mathcal{H}}_2$ are two randomly generated hypervectors ($\vec{\mathcal{H}} \in \{-1, +1\}^D$). The following are the hypervector operations commonly used for implementing HDC-based learning. 1) Binding (*) operation of two hypervectors $\vec{\mathcal{H}}_1$ and $\vec{\mathcal{H}}_2$ is done by component-wise multiplication (XOR in binary) and is denoted as $\vec{\mathcal{H}}_1 * \vec{\mathcal{H}}_2$. The result of the operation is a new hypervector that is dissimilar to its constituent vectors, that is, $\delta\left(\vec{\mathcal{H}}_1 * \vec{\mathcal{H}}_2, \vec{\mathcal{H}}_1\right) \approx 0$; thus, binding is well suited for associating information stored in the two hypervectors. Binding is used for variable-value association and, more generally, for mapping. 2) Bundling (+) operation is done via component-wise addition of hypervectors, denoted as $\vec{\mathcal{H}}_1 + \vec{\mathcal{H}}_2$. The bundling is a memorization function that keeps the information of input data in a bundled vector. The bundled hypervectors preserve similarity to its component hypervectors, i.e., $\delta\left(\vec{\mathcal{H}}_1 + \vec{\mathcal{H}}_2, \vec{\mathcal{H}}_1\right) >> 0$. 3) Similarity measure between two vectors $\vec{\mathcal{H}}_1$ and $\vec{\mathcal{H}}_2$ is defined as: $\delta\left(\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2\right) = \vec{\mathcal{H}}_1^{\dagger} \cdot \vec{\mathcal{H}}_2 / D$. For example, if the two hypervectors are randomly generated, $\delta\left(\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2\right) \approx 0$ because the dot product measures the high-dimensional similarity of the two near-orthogonal hypervectors. If we assume that $\vec{\mathcal{H}}_1$ and $\vec{\mathcal{H}}_2$ are two complex-valued vectors, we can use the operation †, which transposes the column vector and takes the conjugate of all components. This similarity operation gives us a complex scalar value.

3. NetHD Encoding

In this work, we propose a novel encoding scheme that maps an arbitrary bitstream to a high-dimensional space. *NetHD* encoding exploits HDC mathematics to preserve all information of data in an encoded hypervector. **Figure 2** shows the *NetHD* encoding functionality. Let us assume that a bitstream is stored as an array $\vec{S}$ with a length of L ($\vec{S} \in \{0,1\}^n$). Our goal is to map this bitstream into a hypervector $\vec{\mathcal{H}}$ of D dimension. Our encoding occurs using the following steps:

3.1. Chunk Mapping

We divide an input bitstream into V chunks of length L/V each (Figure 2a). Define the ith chunk to be $C_i = S\left[(i-1)\cdot\frac{L}{V} : i\cdot\frac{L}{V}\right]$ for $i = 1, 2, 3, .., V$. We construct a mapping table for every $\frac{L}{V}-$ digit binary vector to represent each with a random hypervector. We denote this mapping table by $\vec{\mathcal{F}}(x)$ where x is a vector of $\frac{L}{V}$ digits. This function maps different chunks to high-dimensional points with a nearly orthogonal distribution, which means that $\delta\left(\vec{\mathcal{F}}(C_i), \vec{\mathcal{F}}(C_j)\right) \simeq 0$ for $i \neq j$. The orthogonality of hypervectors is ensured as long as the hypervector dimension, D, is large enough compared to the number of features, V, in the original data, i.e., $D \gg V$.

3.2. Preserving Position

To differentiate between feature locations, we also associate a random hypervector to each chunk position, i.e., $\left\{\vec{\mathcal{P}}_1, \vec{\mathcal{P}}_2, \cdots, \vec{\mathcal{P}}_v\right\}$, where $\delta\left(\vec{\mathcal{P}}_i, \vec{\mathcal{P}}_j\right) \simeq 0$ for $i \neq j$. These position hypervectors identify the chunk to which the input belongs.

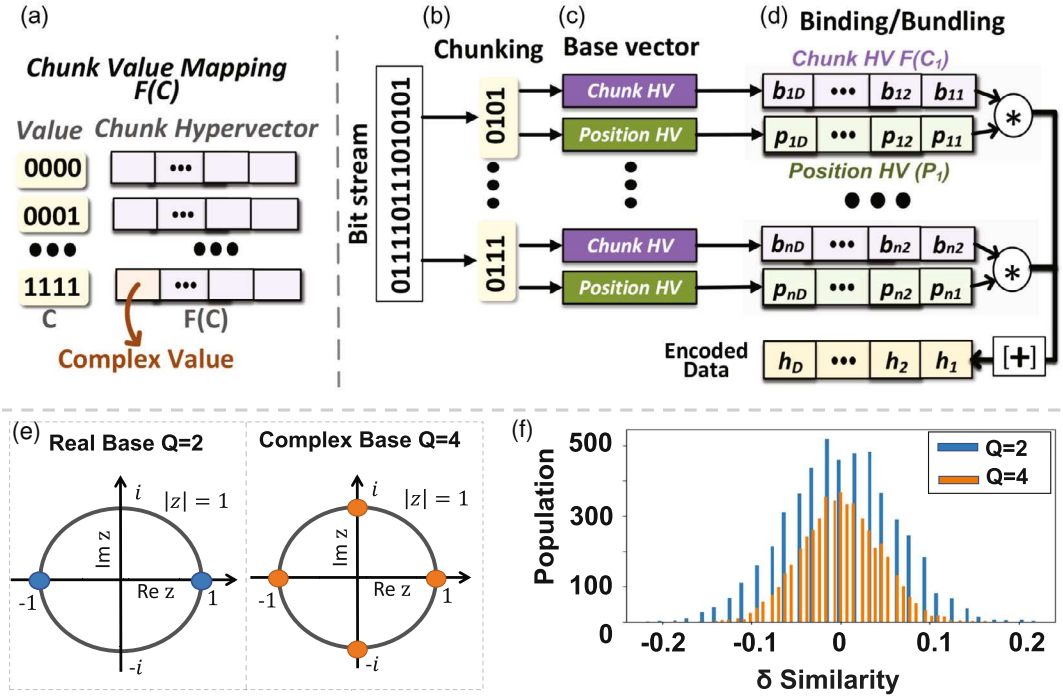

Figure 2. *NetHD* encoding process: a) chunk hypervector generation, b) split a bitstream to small chunks, c) assign a base hypervector to each chunk position, and d) encode the bitstream by associating the position and chunk hypervectors and memorizing them in high dimension. e) Symbol set selection and f) similarity distribution using different real and complex symbols.

3.3. Encoding

We encode the signal by associating each chunk hypervector with the corresponding position hypervector. For example, $\vec{\mathcal{F}}_1 * \vec{\mathcal{F}}(P_1)$ associates the value and position of the first chunk as a new hypervector. The bundling of all associated hypervectors over all chunks memorizes the entire bitstream

$$\vec{\mathcal{H}} = \frac{1}{\sqrt{V}} \sum_{i=1}^{V} \vec{\mathcal{P}}_i * \vec{\mathcal{F}}(C_i) \tag{1}$$

The result of the equation creates a single hypervector $\vec{\mathcal{H}}$, which preserves the value and position information of all chunks with holographic representation and signal normalization. As the encoding spreads information over hypervector elements, a substantial number of bits can be corrupted while preserving sufficient information.

Here, we explain the functionality of *NetHD* using an example. Let us assume a stream of length $L = 16$, $S = 0110111001011000$. We divide this bitstream into $V = 4$ chunks, $C_1 = 0110$, $C_2 = 1110$, $C_3 = 0101$, and $C_4 = 1000$, where each chunk has length $L/V = 4$. We construct a function or lookup table that maps each 4− digit binary number to a randomly generated hypervector ($\left\{\vec{\mathcal{F}}(C_1), \vec{\mathcal{F}}(C_2), \vec{\mathcal{F}}(C_3), \vec{\mathcal{F}}(C_4)\right\}$). Similarly, we generate a position hypervector, $\left\{\vec{\mathcal{P}}_1, \vec{\mathcal{P}}_2, \vec{\mathcal{P}}_3, \vec{\mathcal{P}}_4\right\}$, for each chunk. Using these two bases, we encode our bitstream as

$$\vec{\mathcal{H}} = \frac{1}{\sqrt{4}} \left(\vec{\mathcal{P}}_1 * \vec{\mathcal{F}}(C_1) + \vec{\mathcal{P}}_2 * \vec{\mathcal{F}}(C_2) + \cdots + \vec{\mathcal{P}}_4 * \vec{\mathcal{F}}(C_4)\right) \tag{2}$$

In our example, the encoded hypervectors will have dimensionality ranging from $D = 128$ to $D = 512$.

One method of further optimizing our algorithm is through binding together more chunks of hypervectors, and using the resonator network technique to factorize the binded vectors. For example, we could bind together 3 subchuncks at position 1 as $\overrightarrow{\mathcal{P}_1} * \vec{\mathcal{F}}(C_1) * \vec{\mathcal{F}}(C_2) * \vec{\mathcal{F}}(C_3)$. This will allow us to store more information in an efficient manner without adding much noise. However, the main problem occurs when we bundle together multiple such binded vectors.[24,41] The noise level increases, due to which many factorization techniques fail to converge. Moreover, even increasing the codebook size can result in many factorization methods to fail simply due to the large search space size, without enough dimensions. The optimal trade-off between the number of factors that can be binded versus the number of such terms that can be bundled together is a current topic of intense research.

3.4. Random Hypervector Generation with Complex Bases

Traditionally, HDC randomly chooses binary ($\{0, 1\}$), or polarized vectors ($\{-1, +1\}$) with uniformly distributed components. One key point to keep in the note is that the bundling must be an invertible operation to recover the associations.

For example, assuming $\vec{\mathcal{B}} = \vec{\mathcal{H}}_1 * \vec{\mathcal{H}}_2$, we can recover components using $\vec{\mathcal{B}} * \vec{\mathcal{H}}_1 = \vec{\mathcal{H}}_2$. It mathematically restricts the capacity of HDC vectors due to a lower number of possible hypervectors when using the conventional polarized vectors.[42–44]

In this work, we unleash and expand the capacity of HDC vectors by sending hypervectors with complex phases. The most general and common HD encoding used is the Fourier holographic reduced representation (FHRR), which encodes $d-$ dimensional feature vector $\vec{x}$ into a $D-$ dimensional HD vector $\vec{\mathcal{H}}_x$, such that the similarity between HD vector reproduces an underlying kernel over the data space as follows

$$\delta\left(\vec{\mathcal{H}}_x, \vec{\mathcal{H}}_y\right) = k(\alpha[\vec{x} - \vec{y}]) \tag{3}$$

The encoding can be constructed by choosing a $D\times d$ -dimensional random matrix from a probability distribution $p(\omega)$, where $p(\omega)$ is the Fourier transform of $k(\vec{x})$, and then defining the encoding as $\vec{\mathcal{H}}_x = \exp(i\alpha M\vec{x})$.

Our inspiration for choosing complex bases is the FHRR representation, where we can represent general correlated elements. The correlation between the hypervectors can be controlled by tuning the value of α— if α is small, then the hypervectors are more correlated, and if α is large, then the hypervectors are more orthogonal. In this way, FHRR allows us more flexibility in choosing the encoding process. For example, in cases where we require learning to be performed over the data, we can directly learn over the encoded data by choosing a more correlative encoder. However, as hardware implementations require quantized bits, we quantize the generic phase values components of the hypervector into uniform points along the unit complex circle.

In the generation of random orthogonal hypervectors for the chunk/position mapping, we can choose the vector component to be any complex phase value with a magnitude of 1. If the memory vector is now $\vec{\mathcal{B}} = \vec{\mathcal{H}}_1 * \vec{\mathcal{H}}_2$, the unbinding operation would be given by $\vec{\mathcal{B}} * \overrightarrow{\mathcal{H}_1} = \vec{\mathcal{H}}_2$, where $\overrightarrow{\mathcal{H}_1}$ is the vector with each component of $\vec{\mathcal{H}}_1$ conjugated. It increases the capacity of the random vectors because the possible random vectors increase exponentially with the size of the symbol set. We call the set of possible symbols S. In this work, the set S is mainly chosen to be $\{\pm 1, \pm i\}$. We will study the effect of different sets of hypervector capacity.

Figure 2e shows *NetHD* choices in the selection of polarized or complex bases. Figure 2f presents the similarity distribution of randomly generated hypervectors using bipolar and complex bases. Our results indicate that random complex vectors have a higher chance of orthogonality, thus showing a narrower distribution. In Section 4.2, we show how the orthogonality of complex bases can reduce the cross-interference noise and increase the memorization capacity.

4. NetHD Demodulation

4.1. NetHD Decoding

For a given signal $\vec{\mathcal{H}}$, *NetHD* uses an iterative decoding to reconstruct the bitstream that successively cancels the predicted noise and attains more accurate guesses.[41,45] In the first iteration, we find the guess values of the chunks $C_i^{(0)}$ by binding the encoded hypervector with the position hypervector (**Figure 3**a)

$$\overrightarrow{\overline{\mathcal{P}_k}} * \vec{\mathcal{H}} = \left(\underbrace{\overrightarrow{\overline{\mathcal{P}_k}} * \vec{\mathcal{P}}_k}_{1}\right) * \vec{\mathcal{F}}(C_k) + \sum_{i=1}^{N}\left(\underbrace{\overrightarrow{\overline{\mathcal{P}_k}} * \overrightarrow{\mathcal{P}_i}}_{\text{Noise}\simeq 0}\right) * \vec{\mathcal{F}}(C_i) \tag{4}$$

This equation gives us a noisy estimate of $\vec{\mathcal{F}}(C_k)$. We use this estimation to recover actual chunk original value, using

$$C_i^{(1)} = \arg\max_{\forall C} \mathrm{Re}\left(\delta\left(\vec{\mathcal{F}}(C), \overrightarrow{\overline{\mathcal{P}_i}} * \vec{\mathcal{H}}\right)\right) \tag{5}$$

As shown in Figure 3b, this equation searches through prestored lookup table entries to find a chunk hypervector that has the highest similarity to our noisy estimation. The search is performed using dot product operation. A lookup table entry with the highest similarity (real part) is our first estimation of the chunk value. This process continues for all chunks to get the first estimation. In Section 6, we explain how this similarity search can be simplified to Hamming distance computation and accelerated in hardware.

We exploit all estimated chunk hypervectors to reduce the noise term in Equation (6). For the nth iteration, the less noisy chunk can be computed using (Figure 3c)

$$\vec{\mathcal{H}}^{(n-1)} = \vec{\mathcal{H}} - \frac{1}{\sqrt{V}}\sum_{i\neq j}\vec{\mathcal{P}}_i * \vec{\mathcal{F}}\left(C_i^{(n-1)}\right) \tag{6}$$

We iteratively continue this process to find a better chunk estimation (Figure 3d). For the nth iteration, we find $C_i^{(n)}$ by

$$C_i^{(n)} = \arg\max_{\forall C} \mathrm{Re}\left(\delta\left(\vec{\mathcal{F}}(C), \overrightarrow{\overline{\mathcal{P}_i}} * \vec{\mathcal{H}}^{(n-1)}\right)\right) \tag{7}$$

We repeat the above iterative process until convergence. In Section 7, we show our study for *NetHD* decoding, which converges quickly.

4.2. Noise and Error Recovery

In this work, we normalize the signal vector $\vec{\mathcal{H}}_0$ to $\delta\left(\vec{\mathcal{H}}_0, \vec{\mathcal{H}}_0\right) = 1$. In the signal vector, we may have a complex Gaussian noise vector overlayed $\vec{\mathcal{N}}$, whose magnitude is distributed with a normal distribution with mean 0 and variance $1/n$. The total transmitted signal is given by $\vec{\mathcal{H}} = \vec{\mathcal{H}}_0 + \vec{\mathcal{N}}$. In this case, the SNR is defined as $10\log_{10} n$. The error due to cross-

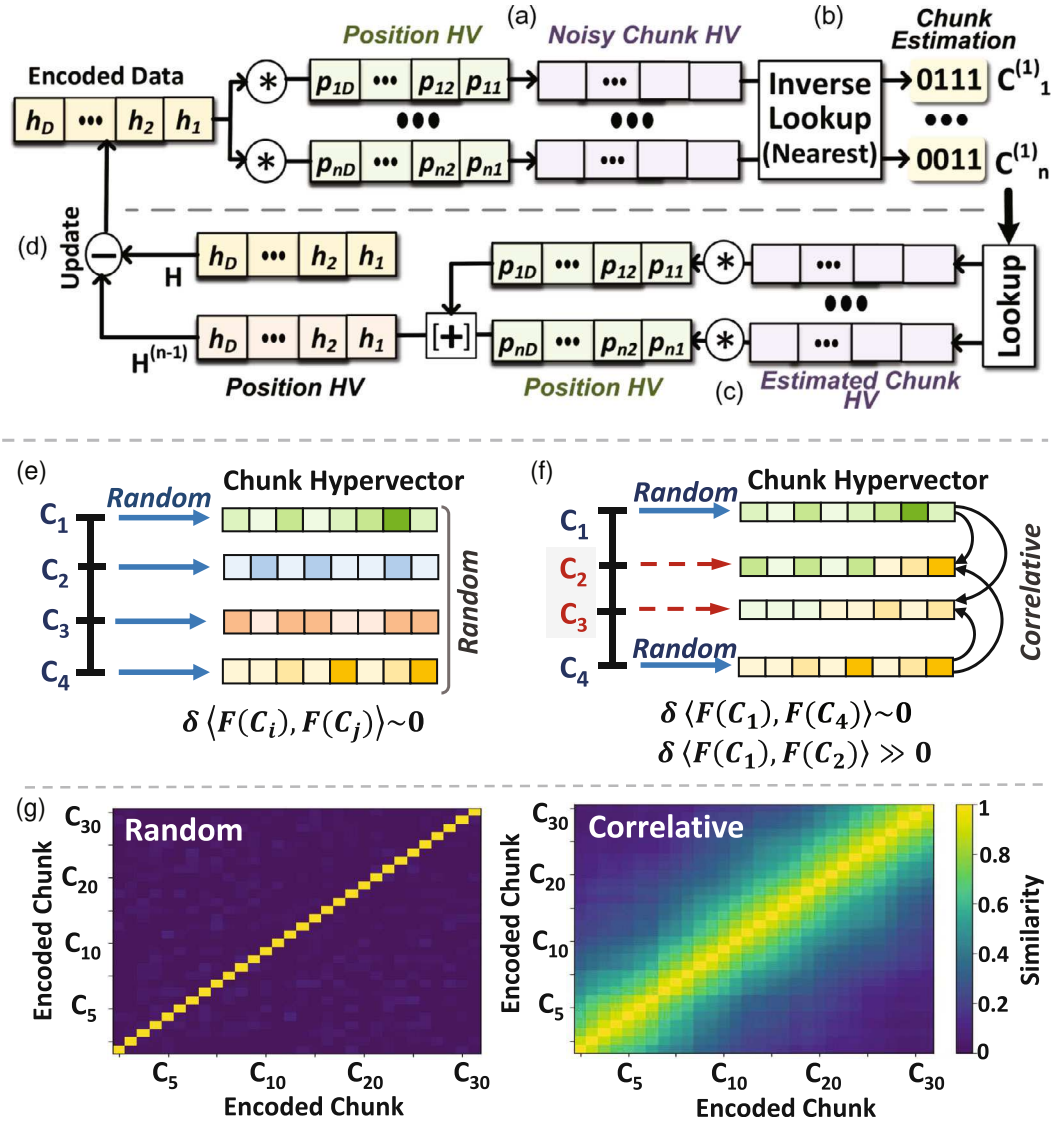

Figure 3. NetHD iterative decoding: a) computes the first estimation of each chunk hypervector, b) mapping estimated hypervector to the closest chunk value, c) exploits the estimated chunk values to re-encoded data, and d) computes the distance of reconstructed data with original encoded hypervector. e) Random base generation used for encoding, f) correlative base generation, and g) visual of correlative mapping of *NetHD* chunks.

interference of the primary terms depends on the dimension of the HD vectors D, the number of layers V and the symbol set used. The error terms would be given by $\frac{1}{\sqrt{V}}\sum_{i\neq j}\vec{\mathcal{P}}_i * \vec{F}\left(C_i^{(n-1)}\right) = \frac{1}{\sqrt{V}}\sum_{i=1}^{V-1}\vec{\mathcal{V}}_i$, where $\vec{\mathcal{V}}_i$ are random uncorrelated vectors. Given a vector representing a value $\vec{\mathcal{R}}$, we are interested in calculating $\mathrm{Re}\left(\delta\left(\vec{\mathcal{R}}, \frac{1}{\sqrt{V}}\sum_{i=1}^{V-1}\vec{\mathcal{V}}_i\right)\right) = \frac{1}{\sqrt{V}}\sum_{i=1}^{V-1}\mathrm{Re}\left(\delta\left(\vec{\mathcal{R}}, \vec{\mathcal{V}}_i\right)\right)$.

The problem is now reduced to estimating the real similarity distribution between two random vectors $\vec{\mathcal{A}}$ and $\vec{\mathcal{B}}$. The similarity can be written as $\delta\left(\vec{\mathcal{A}}, \vec{\mathcal{B}}\right) = \sum_{i=1}^{D}\left(\vec{\mathcal{A}}\right)_i\left(\vec{\mathcal{B}}\right)_i$. Here, $\left(\vec{\mathcal{A}}\right)_i$ denotes the ith component of the vector $\vec{\mathcal{A}}$. Note that if $\vec{\mathcal{A}}$ and $\vec{\mathcal{B}}$ are random with components from the set S, then $\left(\vec{\mathcal{A}}\right)_i\left(\vec{\mathcal{B}}\right)_i$ is also a random element of the set S. The set S is, in general, parametrized by $\left\{e^{\frac{2\pi k i}{Q}}\right\}$, where $k = 0, 1, 2, \ldots, Q-1$ and Q is an integer. The real parts of the set S are given by $\mathbb{S}_r = \left\{\cos\frac{2\pi k}{Q}\right\}$. Thus, $\mathrm{Re}\left(\left(\vec{\mathcal{A}}\right)_i\left(\vec{\mathcal{B}}\right)_i\right)$ is a random element of the set $\mathbb{S}_r$. $\mathbb{S}_r$ has mean $\mu = 0$ and standard distribution of

$$\sigma = \sqrt{\sum_{k=0}^{Q-1}\frac{\cos^2\frac{2\pi k}{Q}}{Q}} \tag{8}$$

As the dimension increases, the real similarity between two random vectors will be distributed as a Gaussian with mean 0 and standard deviation $\frac{\sigma}{\sqrt{D}}$ by the central limit theorem. Thus,

$\mathrm{Re}\left(\delta\left(\vec{\mathcal{A}}, \vec{\mathcal{B}}\right) \sim N\left(0, \frac{\sigma}{\sqrt{D}}\right)\right)$. Therefore, for the more general cases, we have

$$\mathrm{Re}\left(\delta\left(\vec{\mathcal{R}}, \frac{1}{\sqrt{V}}\sum_{i=1}^{V-1}\vec{\mathcal{V}}_i\right) \sim N\left(0, \frac{\sigma}{\sqrt{D}}\right)\right) \tag{9}$$

This equation shows that the contribution from the cross-interference is independent of V. However, note that the term matching $\vec{\mathcal{R}}$ is normalized by the weight of $\frac{1}{\sqrt{V}}$. Thus, the SNR of the cross-terms is given by $10\log_{10}\frac{\sqrt{D}}{\sigma\sqrt{V}}$. Note that σ decreases with increasing Q. Thus, the three ways to decrease noise are by increasing D, increasing Q, and decreasing V. However, each method has its own trade-off. Increasing D reduces the coding rate because a larger number of packets must be transmitted. Increasing Q would make the symbols more closely spaced, resulting in the need for the receiver equipment to distinguish between closely spaced symbols. Decreasing V would increase the size of the chunks, resulting in a larger memory requirement to store all possible bit sequences.

5. Learning in High Dimension

For many IoT applications, the system efficiency depends on both communication and computation, which are separated, unfortunately, in today's systems. For example, to learn the pattern of transmitted data, we still need to pay the cost of iterative decoding. Here, we introduce a solution that incorporates the distance between learning and communication. Instead of paying the cost of iterative data decoding, *NetHD* enables hyperdimensional learning to operate directly on transmitted data, without the need for costly iterative decoding. In particular, we enable HDC classification and clustering of transmitted data with a choice of data compression.

5.1. Learning Encoding

NetHD encoding module maps data points to a high-dimensional space. The goal of this encoder is to represent each data as an orthogonal point, unless they are identical (shown in Figure 3e). This feature is essential for accurate data decoding. However, this encoder is not ideal for some learning tasks. HDC learning fundamentally works by clustering data points that are nonlinearly mapped into high-dimensional space. To simplify data clustering, the encoding module needs to preserve the correlation between input data.

5.1.1. Correlative Bases

As we explained in Section 3, chunk hypervectors, $\vec{\mathcal{F}}(C)$, have been selected to uniquely map each binary vector (chunk) of $\frac{L}{V}$ digits to an orthogonal point in a high-dimensional space. To preserve correlation, our function needs to map physically correlated chunks to similar vectors. We use a quantization method as a map function that generates correlated hypervectors for chunks. As shown in Figure 3f, our map function generates a random hypervector for the first chunk, $\vec{\mathcal{F}}(C_1)$. The rest of the chunk hypervectors are generated by transforming the random dimensions of $\vec{\mathcal{H}}_1$. For example, $\vec{\mathcal{H}}_i$ is generated by flipping $D/2^{L/V+1}$ dimensions of $\vec{\mathcal{H}}_{i-1}$. As HDC learning is approximate, to ease the encoding module, a group of neighboring chunks can also be assigned to a single-chunk hypervector. For example, to represent $L = 8$-bit chunks, we ideally require 256 chunk hypervectors. However, this precision is not required thanks to the statistical nature of ML; thus, we can quantize the chunk values to a much smaller value, for example, 16 or 8 chunk hypervectors.

5.1.2. Data Structured Encoding

Using a new mapping function, we can use the same encoding as in Equation (1). The size of the chunk and the correlation of position hypervectors may change depending on the data structure. For example, if the encoded data correspond to a time series with 8-bit precision values, we can use the chunk size equal to 8-bit. In addition, the position hypervector can be correlated for data with a structure. For example, for time series, the neighbor position hypervectors should have a higher correlation. An important note is that HDC learning will work accurately even with random position hypervectors. Using the correlative position hypervector only decreases the required dimensionality to achieve the maximum quality of HDC learning.

Figure 3g visually shows the similarity of the encoded chunks using random and correlative bases. Each axis shows an encoded 30-bit chunk, where chunks are of the form $C_j = 00..0011..11(31-j$ 0s and j 1s) from $j = 1, 2, 3, .., 30$. We chose this representation because C_j and C_{j+1} would then have exactly a one-bit difference and would be able to demonstrate the similarity of the correlated encoder conveniently. As the heatmap shows, using both random and correlative bases, the diagonal has the highest similarity, indicating that each encoded chunk has full similarity to itself. Our evaluation shows how our correlative mapping keeps the similarity of encoded chunks with a closer physical distance in the original space. In contrast, using random bases, the encoded chunks cannot preserve similarity to any other chunks, even if they are highly correlative (e.g., a single bit difference). Note that *NetHD* can quantize floating point or even complex values into discrete levels that can be a representation of our encoder. For example, a 32-bit floating-point representation can be quantized to 8-bit before encoding.

5.2. NetHD Classification

Training starts with accumulating all encoded hypervectors corresponding to each class. As shown in **Figure 4**b, the result will be k class hypervectors, where k is the number of classes. Assuming there are J inputs that have the label l, the class hypervector is computed by $\vec{\mathcal{C}}_l = \sum_j^J \vec{\mathcal{H}}_j^l$. Retraining examines if the model correctly returns the label l for an encoded query $\vec{\mathcal{H}}$. If the model mispredicts it as label l', the model is updated as follows

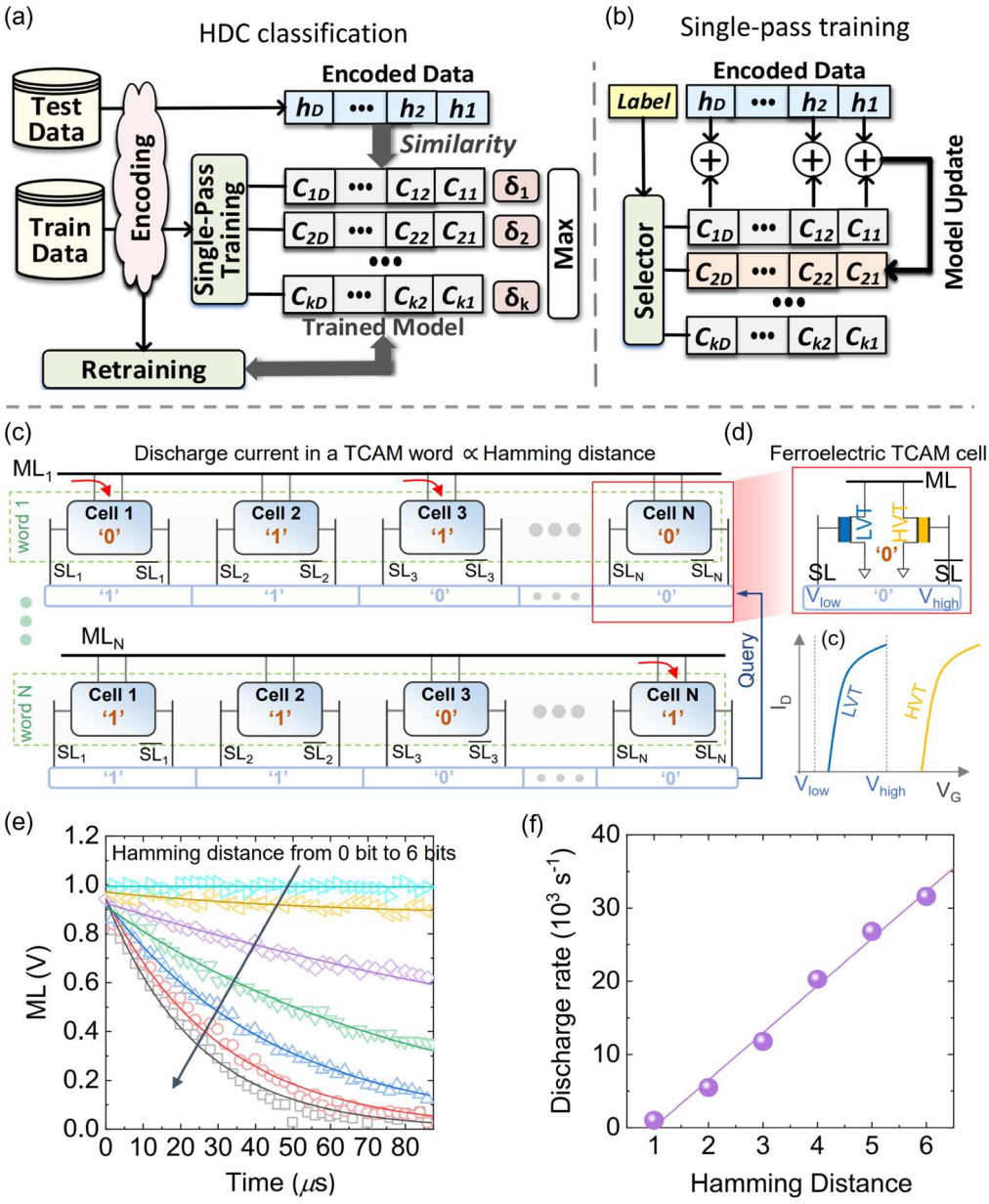

Figure 4. a) Dimensional classification steps, b) single-pass training in hyperspace, and c) CAM array can parallelly calculate the Hamming distance between the query and each stored entry in memory by sensing the discharge current. d) FeFET can realize an ultracompact CAM by utilizing its two V_{TH} states. e) The ML voltage discharge rate increases with the hamming distance. f) The discharge rate is actually proportional to the hamming distance.

$$\vec{C}_l \leftarrow \vec{C}_l + \eta(\delta_{l'} - \delta_l) \times \vec{\mathcal{H}}$$
$$\vec{C}_{l'} \leftarrow \vec{C}_{l'} - \eta(\delta_{l'} - \delta_l) \times \vec{\mathcal{H}} \quad (10)$$

The retraining continues for multiple iterations until the classification accuracy (over validation data) has small changes during the last few iterations.

Inference starts by encoding the test data to a high-dimensional space using the same encoding module used for training. The encoded data are called the query hypervector $\vec{\mathcal{H}}$ (Figure 4a). Next, we compare the similarity (δ) of $\vec{\mathcal{H}}$ and all the class hypervectors to find the class with the highest similarity.

5.3. NetHD Clustering

Clustering is a native functionality supported by high-dimensional models. In high dimension, HDC separates data points while still preserving their correlative distances. This enables low complexity and transparent separation of encoded data points. We exploit the similarity search in high-dimensional space to cluster data points in different centers.

Assume $\vec{\mathcal{H}}$ as a new training data point. *NetHD* generates k random hypervectors as the initial cluster centers in the high-dimensional space. HDC stores original nonbinary clusters ($\vec{C}_i$) and a binarized version ($\vec{C}_i^b$). The encoder module generates both nonbinary ($\vec{\mathcal{H}}$) and binary ($\vec{\mathcal{H}}^b$) hypervectors. Each cluster center is updated using all data points assigned to the center, as well as their corresponding confidence level. After assigning each encoding hypervector $\vec{H}$ of inputs belonging to the center/label l, the center hypervector $\vec{C}_l$ can be obtained by bundling (adding) all $\vec{H}$s. Assuming that there are J inputs that have label l, the cluster update happens using: $\vec{C}_l \leftarrow \vec{C}_l + \sum_j^J \alpha_j \vec{H}_j$, where $\vec{H}_i$ is the encoded query data. All cluster updates are performed over the nonbinary copy of the centers.

5.4. Data Compression

HDC learning is mainly a bundling of encoded hypervectors. This bundling aims to create a compressed and representative model of all train data. In practice, bundling can happen before or after sending the encoded data. However, bundling on the receiver is equivalent to a higher communication cost. Instead, *NetHD* can perform a part of those bundling operations during encoding to ensure holographic and compressed data communication. *NetHD* encoder bundles a batch of data into a single packet and transmits compressed data to a receiver. Without the need for decompression or data decoding, *NetHD* starts learning a model over compressed transmitted data. Although this technique overloads the theoretical capacity of a hypervector, our goal is only to learn the pattern (and not accurately decode the data). *NetHD* learning can preserve a general pattern of compressed data. The data compression rate, or in general the coding rate, creates a trade-off between the accuracy of the learning and the cost of communication. A larger compression reduces the communication cost while it may affect the quality of learning. In Section 7.6, we show that HDC learning is surprisingly robust to data compression.

6. Hardware Acceleration

NetHD decoding and learning involve many nearest search operations that we accelerate by exploiting CAM.[46] During learning, the CAM can store the trained clustering or classification models and use them to compute the distance similarity of a query. In this section, we answer the following questions: (1) How to support the nearest search in CAM and (2) How to deal with vectors with complex components.

6.1. In-Memory Search Operation

The exact search is one of the native operations supported by a conventional CAM. The conventional CAM consists of two memory cells that store complementary values (Figure 4). During a search, the row driver of the CAM block precharges all CAM rows (matchlines:MLs) to supply voltage. The search operation starts by loading the input query into the vertical bitlines (BLs) connected to all CAM rows. The CAM based on ferroelectric field-effect transistor (FeFET) has been shown to be ultracompact, high performance, and energy-efficient, representing an almost ideal solution for the associative search applications.[47] FeFET based on ferroelectric HfO_2 is an emerging memory that exhibits high density, great performance, and superior energy efficiency.[48,49] It replaces a typical high-k gate dielectric with a ferroelectric thin film. By applying a positive/negative gate pulse, the ferroelectric polarization direction can be set to point at the channel/gate metal, setting the FeFET to be low-V_{TH} (LVT)/high-V_{TH} state (HVT), respectively.

As shown in Figure 4d, a CAM cell can be constructed with two FeFETs, where complementary V_{TH} states are stored to encode a bit of information. As an example, a bit "0" is stored as the low-V_{TH} state/high-V_{TH} state for the left/right FeFET, respectively. The query information is encoded as the search voltage applied to the SL and $\overline{SL}$ such that when the search matches the stored information, a negligible discharge current flows through the ML; otherwise, the ML rapidly discharges. For example, the query "0" is encoded as the condition where the $SL/\overline{SL}$ is applied with a low and high voltage such that both FeFETs are cut off, contributing negligible current for the match condition, as demonstrated in Figure 4e. Due to its large ON/OFF ratio, when arranging multiple CAM cells into a CAM word, it is possible to detect the Hamming distance between the query and the stored entry by sensing the discharge current flowing through the ML, as shown in Figure 4c. As each CAM word is independent of each other, parallel calculation of the Hamming distance can be achieved directly inside the memory without the necessity of moving the data around.

Figure 4e,f shows our experimental results. A small FeFET CAM word (i.e., 1×6 array) fabricated on a 28 nm industrial FeFET process has been tested, where all the CAM cells are written in bit "0," and then the query information with an increasing number of mismatched bits is applied. The results capturing the ML voltage waveforms clearly show that the increase in the Hamming distance accelerates the discharge of the ML voltage. The extracted discharge rate is shown to be linearly proportional to the Hamming distance, as shown in Figure 4f, which verifies the functionality of the ferroelectric CAM array.

6.2. Search with Complex Hypervectors

As we explained in Section 3, *NetHD* uses vectors with complex-valued components. Here, we introduce a technique that exploits our CAM block to store complex values and compute distance similarity. Let us assume $Q = q_r + q_c i$ and $A = a_r + a_c i$ as two complex numbers, indicating the single dimension of query and stored CAM pattern. The dot product between these two values defines as

$$C = Q \cdot \overline{A} = (q_r \oplus a_r + q_i \oplus a_i) + (q_i \oplus a_r - q_r \oplus a_i)i \tag{11}$$

Although this similarity involves the inner product between complex numbers, in practice, we only require a real portion of the dot product result. This simplifies the similarity metric to the Hamming distance, where each dimension stores real and imaginary values as two adjacent cells. During the search, the CAM computes the Hamming distance of both real and

imaginary parts and accumulates its result as discharging current on the ML. In other words, using the complex number, we can use the same CAM block with double dimensionality.

7. Evaluation

7.1. Experimental Setup

NetHD has been implemented and evaluated using software, hardware, and system modules. In software, we verified *NetHD* encoding, decoding, and learning functionalities using our C++ implementation. In hardware, we implement *NetHD* on multiple embedded platforms: field programmable gate arrays (FPGA), GPU, and the proposed CAM-based accelerator. For FPGA, we describe the *NetHD* functionality using Verilog and synthesize it using the Xilinx Vivado Design Suite.[50] The synthesis code has been implemented on the Kintex-7 FPGA KC705 Evaluation Kit using 5ns clock frequency. We also create an optimized implementation of *NetHD* on Jetson AGX Xavier.

We perform circuit simulations based on an experimentally calibrated FeFET compact model in ref. [51] We also tested *NetHD* functionality over an experimental and fully functional 2×2 CAM array. We exploit our CAM-based architecture to accelerate the decoding and learning process. For system evaluation, we implement an in-house simulation framework based on NS3[52] to evaluate how *NetHD* performs on distributed learning in internet of things (IoT). The simulation framework evaluates *NetHD* in a hardware-in-the-loop fashion. We use NS-3 to simulate communications on distributed network topologies with diverse network mediums. During the simulation loop, the simulator invokes the *NetHD* learning procedures (wrapped with ApplicationContainer of NS3) on actual platforms that represent different nodes in the IoT hierarchy. *NetHD* is added as the plugin module while testing data are streamed as inputs of sensing nodes within NS3. This allows us to analyze how well HDC can work with missing (lost packets in transmission) or incorrect (bit errors) data.

Table 1 summarizes the practical datasets evaluated for classification. The benchmarks tested consist of large data for smart cities, physical monitoring, and performance/power prediction.

7.1.1. PECAN

This dataset focuses on urban electricity prediction, featuring 312 attributes to model and forecast electricity usage in urban settings, classified into three distinct groups. With 312 end nodes, the dataset provides a robust framework for understanding and predicting energy consumption with a substantial training size of 22 290 instances and a test set of 5,574 instances.

7.1.2. PAMAP2

Tailored for activity recognition using inertial measurement units (IMUs), PAMAP2 contains 75 features across five classes, designed to distinguish between different types of physical activities. The dataset, divided into three end nodes, includes a vast training set of 611 142 samples and a test set of 101 582 samples, offering extensive data for creating detailed activity recognition models.

7.1.3. APRI

Aimed at performance identification, this dataset involves 36 features and two classes, covering a spectrum of performance metrics. With three end nodes, APRI provides a training dataset comprising 67 017 instances and a smaller test set of 1,241 instances, facilitating the development of models to identify and evaluate performance outcomes effectively.

7.1.4. PDP

Designed for power demand prediction, the dataset includes 60 features across two classes, intended for modeling and forecasting power demand. Containing five end nodes, it offers a training set of 17 385 instances alongside a test set of 7,334, enabling detailed analysis and prediction of power consumption patterns.

We also evaluate the quality of *NetHD* clustering on four datasets listed in **Table 2**. To measure cluster quality, we rely on correct labels of data points and find out how many points were classified in a cluster that does not reflect the label associated with the point.

Table 2. Clustering Datasets (*n*: feature size, *K*: # of clusters).

	Data size	*n*	*k*	Description
MNIST	70 000	784	10	Handwritten digit recognition[70]
UCIHAR	10 299	561	6	Human activity recognition[71]
SYNTHET I	1000	100	25	Synthetic data
SYNTHET II	100 000	100	25	Synthetic data

Table 1. Classification datasets (*n*: feature size, *K*: # of classes).

	n	*K*	# End nodes	Train size	Test size	Description
PECAN	312	3	312	22 290	5574	Urban electricity prediction[66]
PAMAP2	75	5	3	611 142	101582	Activity recognition (IMU)[67]
APRI	36	2	3	67 017	1241	Performance identification[68]
PDP	60	2	5	17 385	7334	Power demand prediction[69]

7.1.5. MNIST

A classic benchmark for handwritten digit recognition, MNIST contains 70 000 images with 784 features each, categorized into ten clusters corresponding to the digits 0 through 9. This dataset serves as a prime example for testing clustering algorithms in image classification.

7.1.6. UCIHAR

Utilized for human activity recognition, this dataset includes 10 299 instances with 561 features each, organized into six clusters. These represent different human activities, providing a comprehensive dataset for evaluating clustering in the context of wearable computing data.

7.1.7. SYNTHET I and SYNTHET II

These synthetic data sets are designed to test clustering algorithms under controlled, yet challenging conditions. Both datasets consist of 100 features with SYNTHET I containing 1,000 instances across 25 clusters and SYNTHET II encompassing a larger scale with 100 000 instances also across 25 clusters. They serve as valuable tools for assessing the scalability and effectiveness of clustering methods.

In these evaluations, the focus is on the practicality and efficiency of the *NetHD* clustering method, analyzing how well it groups data points into clusters that accurately reflect the associated labels, thereby providing insight into the algorithm's capability to handle real-world data complexities.

NetHD has primarily three parameters: the size of the chunk C, the dimension D, and the number of layers V. The chunk size is the number of bits encoded in each layer. The D denotes the number of channels being transmitted (dimensions), and V denotes the number of layers encoded in a single series transmitted. The total number of bits being transmitted is $C \times V$, and so the coding rate is given by $R = C \times V/D$. For example, in our typical setting, each layer would transmit $C = 8$ bits of information. If we chose the number of layers to be $V = 8$ and the dimension to be $D = 128$, then the coding rate is equal to $R = 64/128 = 0.5$.

7.2. Bit Error Rate and Noise

We report various bit error rates for arbitrary bitstreams as a function of dimensions D, layers V, and SNR (dB). **Figure 5**a shows the decoding accuracy of *NetHD* as a function of dimension and layers for three different SNR values: -3, 0, and 5 dB. Regardless of the number of layers and SNR values, the decoding accuracy increases with the dimensionality of the channel. This is due to the increasing pseudo-orthogonality of random hypervectors in high-dimensional space. In other words, the dimensionality increases the chance of randomly generated chunk hypervectors to have distinct and orthogonal distribution, thus decreasing the noise from cross-interference terms in Equation (6).

As explained in Section 4.2, each hypervector has a limited capacity to memorize information. Increasing the number of layers, V, lowers the coding rate, as the transmitted hypervector stores more chunk hypervectors. This increases the number of terms that contribute to cross-interference noise during iterative content recovery. As a result, our iterative data decoding can yield lower accuracy. Figure 5a also shows that a lower SNR value can increase the relative magnitude of the noise. This causes errors in the recovery cycle, which causes a higher bit error rate. In a fixed number of layers, *NetHD* with low SNR requires a higher dimensionality to ensure highly accurate data decoding. For example, for $V = 8$, *NetHD* requires a dimensionality of 256, 128, and 64 to ensure a fully accurate data decoding for SNR of -3 dB, 0 dB, and 5 dB.

Figure 5b shows the decoding accuracy as a function of a number of layers, V, and SNR at $D = 128$. Increasing the SNR makes the main signal stronger, which reduces the cross-correlation errors in the iterative decoding method. Therefore, it results in an increase in the decoding accuracy. Similarly, a larger number of layers, V, increases the cross-correlation noise and *NetHD* decoding accuracy. Figure 5b also shows the decoding accuracy as a function of dimension and SNR for $V = 6$ layers. With a larger D value, it increases the chance of orthogonality of randomly generated hypervectors. It translates to lower decoding noise during data recovery.

7.3. Decoding Iterations

Figure 5c shows the number of iterations required for convergence as a function of SNR for different *NetHD* configurations. The results are the average number of iterations reported for 20 evaluation runs. *NetHD* with large SNR has smaller noise; this can accurately decode data with a lower number of iterations. As the SNR decreases, the transmitted signal gets higher noises from the network; thus, *NetHD* requires more iterations for data decoding. As shown in Figure 5c, *NetHD* requires the maximum number of iterations for SNRs in the range of -2 to $+2$ dB. Lowering the SNR below -2 dB, *NetHD* decoding can recover information with a few iterations. This happens as *NetHD* cannot ensure accurate data decoding in a highly noisy network. This causes the converged decoded signal to be high error and random, which requires a lower number of iterations to attain. As a result, we observe higher variations in the number of iterations as the signal strength decreases. *NetHD* with a larger number of layers gets higher cross-interference noise, resulting in a higher variation and the number of decoding iterations.

Figure 6a,b visually shows *NetHD* quality of decoding during different decoding iterations and using the network with various SNR values ($V = 6$, $D = 64$). Our result indicates that, regardless of the SNR value, the decoding accuracy of *NetHD* increases with the number of iterations. However, the decoding accuracy at the final iteration can still be imperfect when an image is transmitted over a low-SNR network. For example, a decoded image under SNR $= -2$dB has a small amount of noise, while an image can be perfectly decoded under SNR $= 0$ dB.

7.4. NetHD Versus State-of-the-Art

We compare the decoding accuracy of *NetHD* with the state-of-the-art HDM.[13] Figure 6c shows the difference between *NetHD*

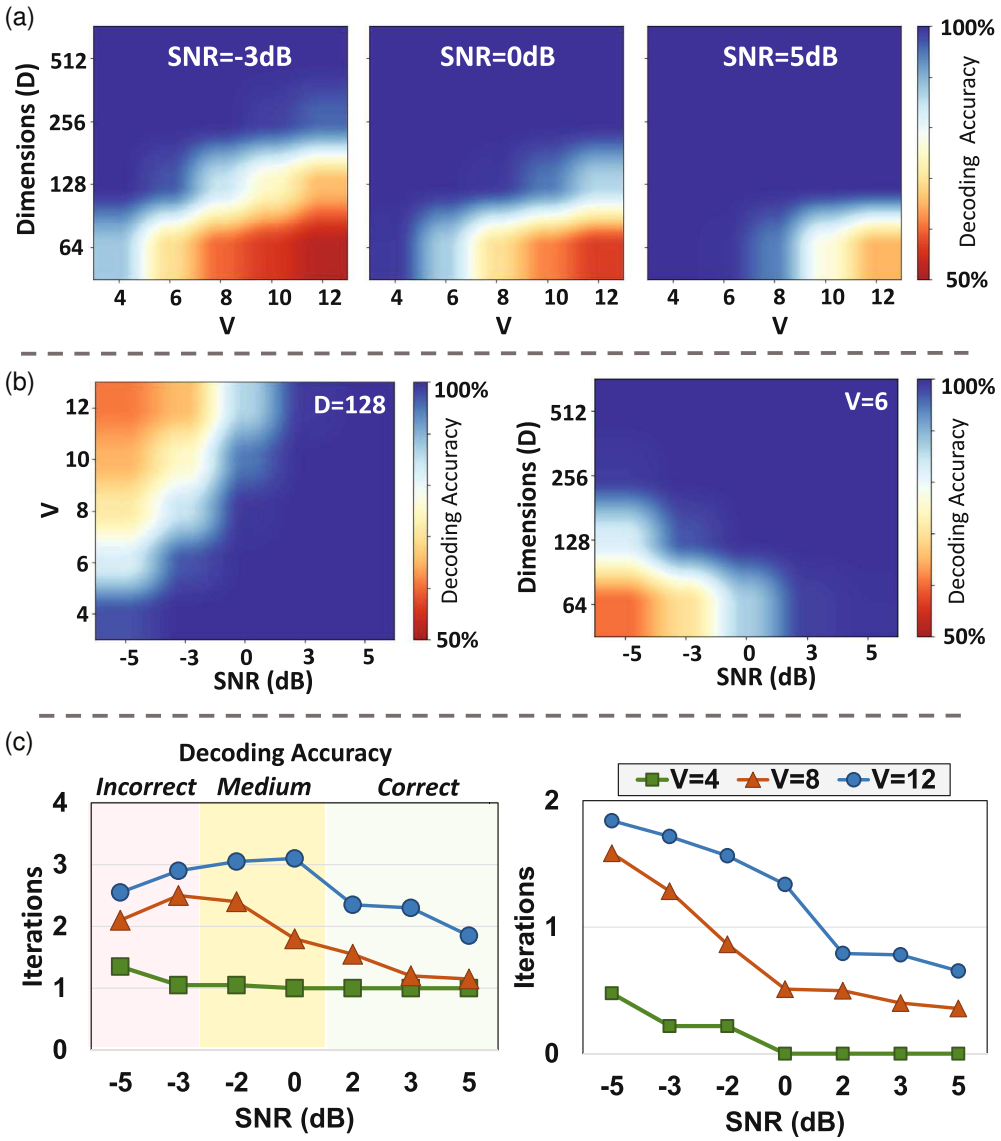

Figure 5. a) *NetHD* decoding as a function of number of layers and dimensionality. b) *NetHD* iterative decoding: a) average number of iterations, b) *NetHD* decoding in fixed dimensionality and the number of layers, and c) *NetHD* iterative decoding (left) average number of iterations, (right) standard deviation over 20 experiments.

and HDM decoding accuracy under various SNR values and using a different number of layers. All results are reported when both *NetHD* and HDM have the same coding rate ($R = 0.5$). The results indicate that in most configurations, *NetHD* outperforms HDM in terms of decoding accuracy, specifically in regions of low noise and a small number of layers. As we showed in Figure 6, *NetHD* has a 100% accuracy in these regions, while the HDM enables approximate decoding. This is because the HDM model encodes the vectors in a nonrandom way, while our model generates all the lookup bases randomly, thus ensuring pseudo-orthogonal chunk representation. As a result, our model essentially does an exact search over all the layers, resulting in perfect data decoding.

The HDM accuracy of decoding is better than *NetHD* under conditions of low SNR and a large number of layers, resulting in large noise in the iterative decoding step. In these configurations, *NetHD* has a higher vulnerability, as noise can modify similarity so that two different random vectors that might have larger similarities can be confused with each other. In addition, *NetHD* fundamentally works based on the nearly orthogonal distribution of patterns in a high-dimensional space. In low-dimensional space, the vector cannot ensure the orthogonality of hypervectors, thus increasing the cross-interference noise. As shown in Figure 6c, for low SNR signals, *NetHD* should use a larger dimensionality to reduce the impact of interference noise, thus improving the quality of decoding. Similarly, using a large number of layers, *NetHD* requires a higher dimensionality to ensure that the capacity of an encoded hypervector does not exceed the V value (shown in Figure 6d).

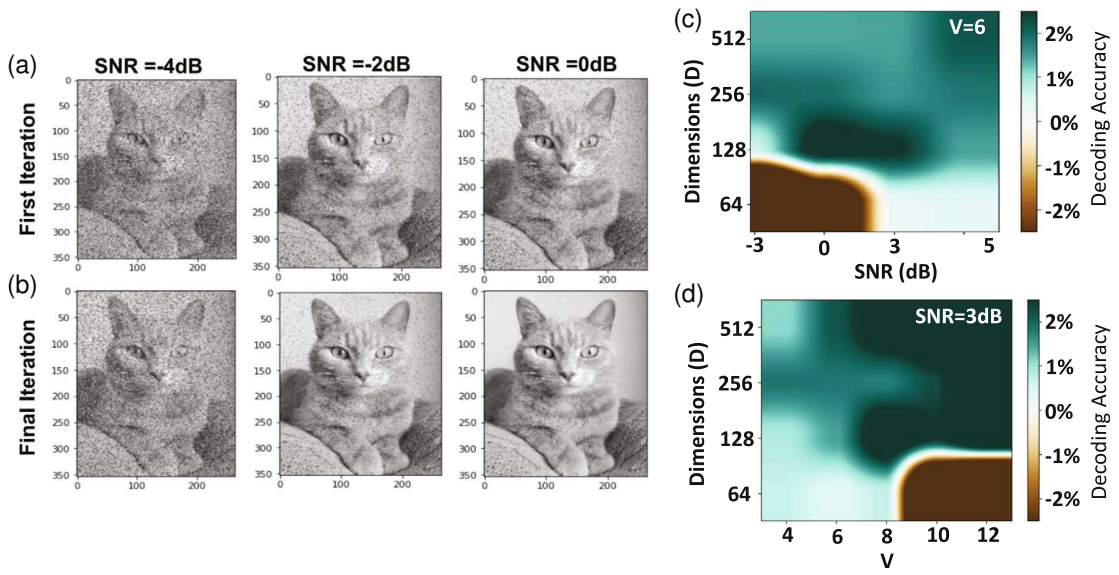

Figure 6. a,b) *NetHD* visual decoding during different decoding iterations and under different SNR values. c,d) Difference in decoding accuracy between NetHD and the state-of-the-art modulation. In green regions NetHD outperforms state-of-the-art.

7.5. NetHD Learning Accuracy

7.5.1. NetHD Learning Accuracy

Figure 7a compares *NetHD* classification accuracy with the state-of-the-art classification algorithms, including DNN in TensorFlow,[53] support vector machine, and AdaBoost on Scikit-learn.[54] The results are reported when all algorithms are performing in a central node that considers all features given in the dataset. We exploit the common practice of grid search to identify the best hyperparameters for each model. Our evaluation shows that *NetHD* provides accuracy comparable to state-of-the-art learning solutions while operating over noisy encoded data.

Figure 7a also shows *NetHD* quality of clustering with the state-of-the-art clustering approaches: *k*-means and locality sensitive hashing (LSH cluster)[55] that clusters data after mapping data into high-dimensional space. K-means algorithm works on original data and uses the Euclidean distance as a similarity metric. Other approaches map data points to dimensions $D = 4k$ before clustering. For LSH- and HDC-based clustering, the results are reported using both cosine metrics. Our evaluation shows that *NetHD* provides a clustering quality comparable to k-means, which is significantly higher than the LSH-based approach.

7.5.2. Coding Rate

We also compare *NetHD* accuracy in different configurations. *NetHD* accuracy depends on both dimensionality and the number of chunks. An increase in dimensionality improves hypervector capacity, thus resulting in a higher quality of learning. On the other hand, increasing the number of chunks results in higher data compression by storing more encoded data in each class hypervector. As explained in Section 7.2, to ensure nearly accurate data decoding, the coding rate should be around $R = 0.5$ or lower. However, learning algorithms are approximate and do not require to ensure accurate data decoding. Our results indicate that *NetHD* can enable accurate learning of highly compressed data with a high coding rate. The high robustness of *NetHD* learning to compression comes from two factors: (1) data compression is holographic, where compressed data sufficiently memorize the information of the individual encoded data. (2) The compression uses the same bundling operation used for model training. Our evaluation indicates that *NetHD* can ensure maximum classification accuracy using 16× smaller data ($R = 8$). Even aggressive model compression of 32× ($R = 16$) and 64× ($R = 32$) only adds 0.7% and 3.9% quality loss to HDC classification.

As shown in Figure 7a (right), the HDC clustering algorithm has a robustness similar to that of data compression. Clustering a batch of encoded data results in the generation of a model with similar quality to baseline clustering. Our evaluation shows that *NetHD* ensures no quality loss (less than 0. 4%) for clustering with 16× (32×) data compression, thus resulting in a significant reduction in data communication.

7.6. NetHD Efficiency

7.6.1. NetHD Learning Efficiency

Figure 7b compares *NetHD* training efficiency with DNN in different configurations. The results are reported for both FPGA and Jetson Xavier. The results include both communication cost and computation cost. Although DNN training always performs on decoded data in the receiver, *NetHD* learning can be performed in two configurations: 1) NetHD after decoding: transmitted data first become demodulated, and then we perform the learning task over the data and 2) NetHD after encoding: directly learning over transmitted data without need for decoding. Our evaluation shows that *NetHD* without decoding (after

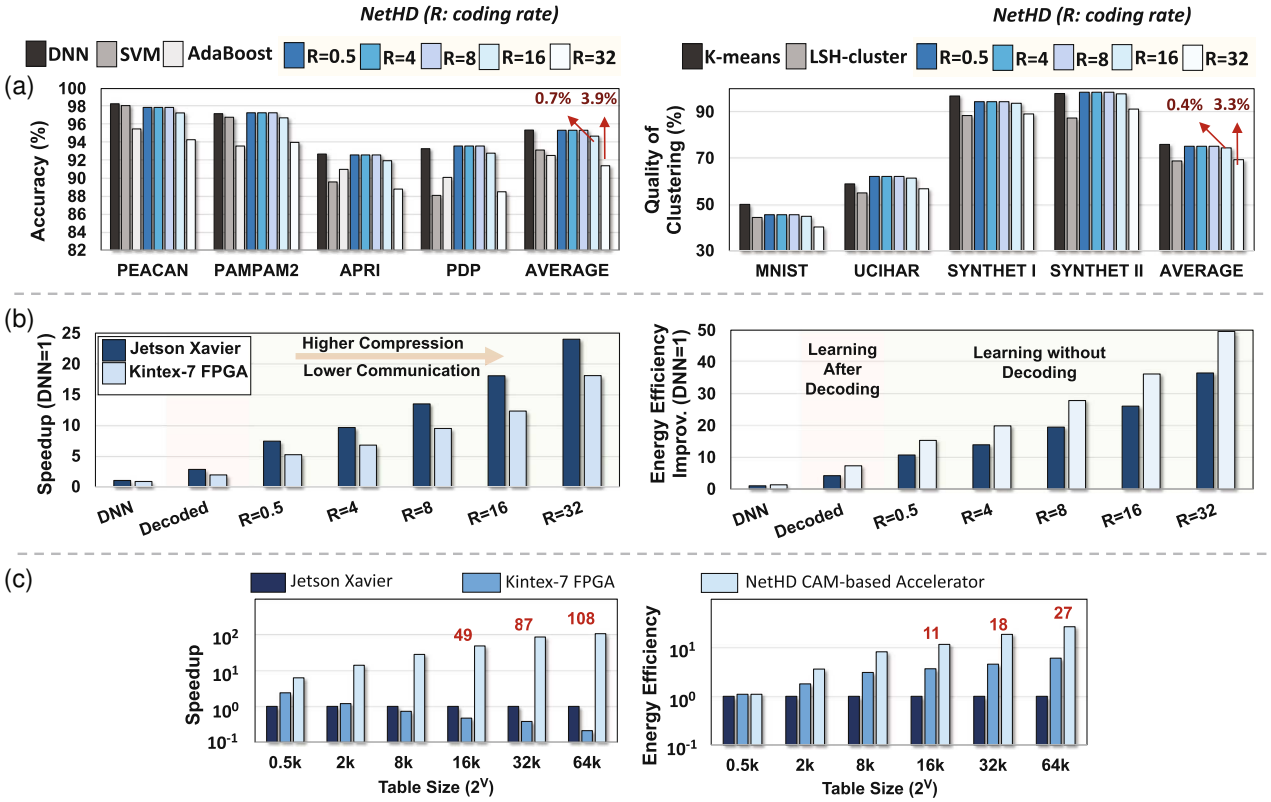

Figure 7. a) *NetHD* quality of classification (left) and clustering (right) versus state-of-the-art and during different coding rate, b) *NetHD* learning speedup and energy efficiency in different configurations and over different platforms (normalized to DNN running on Xavier), c) NetHD decoding efficiency on different platforms.

decoding) can provide 7.3× and 10.5× (2.8× and 4.2×) faster and higher energy efficiency compared to DNN. As *NetHD* learning without decoding eliminates significant computational overhead, we observe higher efficiency.

7.6.2. Efficiency with Compression

Figure 7b shows the impact of coding rate (data compression) on *NetHD* learning efficiency. The efficiency values are averaged among the classification and clustering applications. Our evaluation shows that the learning efficiency of *NetHD* improves with increasing coding rate. This efficiency comes from: 1) a larger coding rate reduces the communication cost by transferring more compressed information through the network. The reduction in communication cost is linear with the coding rate. 2) A high coding rate also improves learning efficiency because HDC models can be trained using fewer train data in compressed form. *NetHD* FPGA using $R = 8$ ($R = 32$) results in 5.1× and 3.8× (9.5× and 6.7×) speedup and energy efficiency improvement, respectively, compared to baseline *NetHD* operating on decoded data.

7.6.3. NetHD Efficiency Under Different Network Conditions

Table 3 explores the impact of network bandwidth on *NetHD* computational efficiency. We have evaluated the efficiency of *NetHD* performance on five network mediums: a wired network of 1 Gbps, a wired network of 500 Mbps, WiFi 802.11ac, WiFi 802.11n, and Bluetooth 4.0. The results for Jetson Xavier are reported using $R = 8$, ensuring no quality loss. The results show that when the network bandwidth is more limited, *NetHD* achieves higher speedup. For example, using 802.11ac with 46.5 Mbps, *NetHD* achieves on average 4.8× speedup compared to *NetHD* operating over decoded data. This speedup increases to 18.1× when we use even lower bandwidth networks such as Bluetooth 4. In practical IoT systems, the bandwidth of the network can usually be limited. The recent embedded devices, i.e.,

Table 3. *NetHD* efficiency using different network bandwidth.

	Bluetooth	802.11n	802.11ac	500 Mbps	1 Gbps
Speedup	13.2×	6.7×	4.8×	3.4×	1.9×
Energy efficiency	18.1×	7.9×	4.4×	3.1×	1.7×

Table 4. *NetHD* efficiency on different platforms.

	Speedup		Energy efficiency	
	Classification	Clustering	Classification	Clustering
GPU	14.8×	10.3×	21.4×	15.5×
FPGA	11.3	6.7×	28.9×	20.3×
CAM	183.7	242.4×	338.4×	448.6×

Raspberry Pi 3 Model B+, uses WiFi 802.11ac and Bluetooth 4.0, which practically provide 23.5 Mbps and 1 MBps bandwidth, respectively. Therefore, *NetHD* is a suitable solution for IoT systems that typically have limited bandwidth.

7.7. NetHD Acceleration on CAM

NetHD decoding and learning rely on highly parallel nearest search operations, which can be significantly accelerated using CAM blocks. To ensure scalability, we limit our CAM size to 1k rows, and for configurations that require larger hypervectors, we exploit multiple CAM blocks for a parallel search. Our evaluation in Figure 7c shows that our CAM outperforms the GPU and FPGA in terms of decoding speed and energy efficiency. The CAM capability of *NetHD* in offloading the search operation is the key to its efficiency, which increases depending on the chunk size. Our evaluation demonstrates that the CAM accelerator provides faster and more energy-efficient decoding than GPU/FPGA, especially for larger chunk sizes. For example, our CAM provides 108.3× and 27.1× (247.4× and 4.5×) faster and more energy efficient than GPU (FPGA).

Table 4 summarizes the improvements in performance acceleration and energy efficiency for learning using CAM with $R = 16$. All results are normalized to the execution time and energy consumption of DNN running on Jetson Xavier. Our results indicate that the CAM accelerator can significantly enhance *NetHD* computation efficiency by accelerating costly associative search. Our evaluation shows that *NetHD* clustering achieves 23.4× and 28.8× (35.8× and 22.0×) faster and higher energy efficiency as compared to *NetHD* running on Jetson Xavier (FPGA), respectively.

8. Related Work

Prior research have applied the idea of HDC to diverse cognitive tasks, such as robotics,[23,56] latent semantic analysis,[57] language recognition,[58] gesture recognition,[59] biosignal processing,[60,61] and distributed sensors.[62,63] Several recent works have focused on designing a novel hyperdimensional encoding for different data types.[25,64] However, the encoding methods used in the previous work are mostly for specific data types and learning applications. In contrast, *NetHD* introduces a general encoding scheme that deals with any arbitrary bitstream while preserving spatial–temporal information.

Recent research in HDC mainly focused on the classification task, aiming to design HDC learning modules for low-power embedded devices.[22,34,65] However, all existing solutions consider the computation power, while communication often dominates the entire energy consumption in IoT systems with low-bandwidth networks.[21] On the other hand, HDM is developed for ultrareliable low-latency communication.[13,17,18] HDM already showed higher reliability than BPSK, LDPC, Polar, and convolutional codes. However, there are multiple challenges with existing HDM modulations: 1) HDM decoding or demodulation is a costly iterative process that involves an extensive search for noise cancellation. 2) the HDM is only focused on modulation and does not get the benefits of HDC. On the contrary, *NetHD* introduces an iterative demodulation technique that uses a hardware accelerator for fast and reliable data decoding. Unlike prior work, which focused only on communication, *NetHD* fundamentally merges HDM and learning to maximize the benefit with a new demodulation technique and hardware design.

9. Feasibility and Future Directions

To study the feasibility and scalability of NetHD in large-scale systems, a comprehensive approach is needed. First, evaluating NetHD's performance under various network conditions, including fluctuating SNRs and bandwidth constraints, is crucial. Understanding its resilience to different types of interference and adaptability to changing network environments ensures reliability and consistent performance. Additionally, assessing how NetHD integrates with existing network infrastructures and protocols is vital. It is important to determine the necessary hardware and software modifications for seamless deployment and identify potential compatibility issues with current systems. This helps in facilitating smooth integration without substantial overhauls to existing infrastructure. As our model works with simple low precision operations, it is naturally robust against noise as we show in our evaluations.

Second, conducting detailed performance benchmarks against state-of-the-art systems is essential for gauging NetHD's efficiency. Comparisons should be made in terms of processing speed, energy consumption, and error rates across various hardware platforms. Investigating the trade-offs between data transmission rates, accuracy, and computational demands, particularly focusing on the effects of data compression and coding rates, will help in optimizing NetHD for different use cases.

Furthermore, NetHD's learning and adaptation capabilities warrant thorough examination. Its efficacy in processing and classifying high-dimensional data across diverse applications needs to be evaluated. Additionally, its ability to adapt to new data patterns and update its models in response to changing environmental inputs or requirements is critical for maintaining relevance and utility in dynamic settings. Like we show in our work, and many other work, HDC can easily learn information from large dimensional data using kernel encodings.

Security and privacy also pose significant concerns, especially given the high-dimensional nature of the data NetHD handles. Strategies must be developed to ensure the integrity and confidentiality of transmitted data, safeguarding against unauthorized access or tampering. This involves assessing potential vulnerabilities and implementing robust security measures tailored to HDC environments. Due to the large dimensional random encoding nature of HD algorithms, it can be often hard to decode them without access to the random matrix or the codebook.

Finally, understanding the diversity of users and devices that NetHD will serve is imperative. The system must cater to a wide range of devices, from low-power IoT sensors to high-end servers, and meet varied application-specific requirements. This ensures that NetHD delivers optimal performance and energy efficiency across different scenarios. Moreover, our designs can be directly implemented in low power microcontrollers like Arduinos and Rasberry Pi, though to utilize the full hardware efficiency of our algorithms, it is better to equip the sensors with specialized HD softwares. Additionally, the long-term viability, maintenance, and upgrade paths for NetHD should be considered, alongside economic and logistical aspects, to ensure its practical deployment and sustained operation in large-scale systems. Addressing these areas will provide a holistic understanding of NetHD's deployment potentials and practical applications.

10. Conclusion and Discussion

In this article, we introduced NetHD, a novel framework that leverages HDC to enable robust and efficient data communication and learning. NetHD integrates data modulation and learning processes by exploiting the redundant and holographic representation of HDC vectors. NetHD encodes data into high-dimensional vectors that can be transmitted through noisy channels with high reliability. NetHD also enables various HDC-based learning tasks to be performed directly over the encoded data without costly decoding. Moreover, NetHD introduces a complex-valued representation for HDC vectors that enhances the capacity and orthogonality of the encoding scheme. NetHD also designs a hardware accelerator based on ferroelectric memory devices that supports fast and efficient data decoding and learning using CAM.

NetHD addresses the challenges and opportunities of next-generation communication systems, such as 6G networks, that require ultralow latency, high reliability, and AI integration. NetHD offers several advantages over existing solutions, such as:

Providing comparable bit error rate to state-of-the-art modulation schemes while fundamentally merging HDM and learning. Unlike conventional modulation schemes that require separate modules for data encoding, decoding, and learning, NetHD unifies these steps into a single pipeline by using HDC vectors as both data carriers and feature representations for learning. NetHD also achieves a bit error rate similar to or lower than existing modulation schemes, such as LDPC, Polar, and HDM, by exploiting the error correction capability of HDC vectors.

Achieving significant speedup and energy efficiency improvements compared to state-of-the-art deep learning systems for both data communication and learning tasks. This is because HDC operations use simple arithmetic functions that can be efficiently implemented on the ferroelectric hardware.

Enabling lightweight privacy and security by exploiting the holographic and compressed representation of HDC vectors. NetHD encodes data into high-dimensional vectors that are distributed over multiple symbols and channels. This makes it difficult for eavesdroppers or adversaries to intercept or tamper with the transmitted data without knowing the encoding parameters and the symbol set. NetHD also compresses the data into lower dimensional vectors that preserve only the essential information for learning tasks. This reduces the exposure of sensitive or personal data to potential privacy breaches or attacks.

NetHD is scalable and adaptable to different network conditions, data types, and learning objectives by tuning the encoding parameters and the symbol set. NetHD allows users to adjust the encoding parameters, such as the dimensionality, sparsity, or orthogonality of HDC vectors, to optimize the trade-off between performance, accuracy, and efficiency. NetHD also enables users to select different symbol sets, such as binary, complex-valued, or frequency domain symbols, to suit different channel characteristics, data formats, or modulation schemes. NetHD can also support different learning objectives, such as supervised, unsupervised, or reinforcement learning, by using different HDC operations or algorithms.

Moreover, we additionally introduce the correlative HDC encoder, which can be used for noisy but holistic signal encoding in cases where the data share some amount of correlations. By using base hypervectors that are correlated, the resulting encoding ensures that correlated data points are encoded into similar data points at the cost of noise decoding. The advantage of this process is that the encoded hypervectors can be fed directly into different deep learning frameworks without going through decoding, thus providing added privacy toward the original signal values.

Some drawbacks of our work are that HDC works best on accelerated hardware, which makes full use of the limited precision arithmetic required to minimize the number of operations and lower energy usage. These hardware devices are not commonly available for use in low power sensor devices and communication nodes. Therefore, one direction of future work is trying to accelerate HDC algorithm on traditional hardware (where HDC is still faster than traditional deep learning methods). Moreover, another challenge is in extending the algorithm for longer data streams and to perform more accurate decoding with correlated representations because this will allow us to compress larger data sizes within the same HD space. In large-scale systems, one aspect to be studied is studying the effects of cross-network interference, where multiple HDC nodes are broadcasting which can cause interference in the symbols received. Therefore, another direction is designing methods for coordinated communications and synchronization within the HDC transmission framework (e.g., methods inspired by ALOHA or Paxos).

We have evaluated NetHD using various datasets and benchmarks and demonstrated its performance, accuracy, and efficiency. We have also implemented NetHD on multiple embedded platforms, such as FPGA, GPU, and CAM-based accelerators, and showed its hardware feasibility and benefits. We believe that NetHD is a promising technology that can revolutionize IoT efficiency by providing a bioinspired framework for intelligent and selective sensor data transmission, and will soon find usage in multiple domains in the future.

Acknowledgements

This work was supported in part by DARPA Young Faculty Award, National Science Foundation #2127780, #2319198, #2321840, and #2312517, Semiconductor Research Corporation (SRC), Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force

Office of Scientific Research under award #FA9550-22-1-0253, and generous gifts from Xilinx and Cisco.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

P.P. and M.I. conceived the research. P.P., Y.Z., Z.Z., K.N., and M.I. conducted the research and analyzed the data. All authors wrote and reviewed the manuscript and agreed on the content of the article.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Keywords

artificial intelligent, channel computing, energy efficiency, hyperdimensional computing

Received: December 4, 2023
Revised: March 31, 2024
Published online: May 26, 2024

[1] Samsung Research, *6G: The Next Hyper Connected Experience for All*, Codeground **2020**.
[2] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, Y.-J. A. Zhang, *IEEE Commun. Mag.* **2019**, *57*, 84.
[3] Y. Zhao, G. Yu, H. Xu, arXiv:1905.04983, **2019**.
[4] H. Viswanathan, P. E. Mogensen, *IEEE Access* **2020**, *8* 57063.
[5] C. Yizhan, W. Zhong, H. Da, L. Ruosen, in *2020 International Conf. on Computer Communication and Network Security (CCNS)*, IEEE, Piscataway, NJ **2020**, pp. 59–62.
[6] S. Ilager, R. Muralidhar, R. Buyya, in *2020 IEEE Cloud Summit*, IEEE, Piscataway, NJ **2020** pp. 1–10.
[7] M. Iansiti, K. R. Lakhani, in *Competing in the Age of AI: Strategy and Leadership When Algorithms and Networks Run the World*, Harvard Business Press, Brighton, MA **2020**.
[8] W. Sun, J. Liu, Y. Yue, *IEEE Network* **2019**, *33*, 68.
[9] P. Yang, Y. Xiao, M. Xiao, S. Li, *IEEE Network* **2019**, *33*, 70.
[10] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, H. Dai, *IEEE Commun. Surv. Tutor.* **2018**, *20*, 3098.
[11] B. Montazeri, Y. Li, M. Alizadeh, J. Ousterhout, in *Proc. of the 2018 Conf. of the ACM Special Interest Group on Data Communication*, Association for Computing Machinery, New York, NY, United States **2018**, pp. 221–235.
[12] J. Sachs, G. Wikstrom, T. Dudda, R. Baldemair, K. Kittichokechai, *IEEE Network* **2018**, *32*, 24.
[13] H.-S. Kim, in *2018 IEEE International Conf. on Communications (ICC)*, IEEE, Piscataway, NJ **2018**, pp. 1–6.
[14] S. Yi, Z. Hao, Z. Qin, Q. Li, in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, IEEE, Piscataway, NJ **2015**, pp. 73–78.
[15] F. Tang, Y. Kawamoto, N. Kato, J. Liu, *Proc. IEEE* **2019**, *108*, 292.
[16] I. Tomkos, D. Klonidis, E. Pikasis, S. Theodoridis, *IT Professional* **2020**, *22*, 34.
[17] C.-W. Hsu, H.-S. Kim, in *GLOBECOM 2020-2020 IEEE Global Communications Conf.* IEEE, Piscataway, NJ **2020** pp. 1–6.
[18] C.-W. Hsu, H.-S. Kim, in *2019 IEEE Global Communications Conference (GLOBECOM)*, IEEE, Piscataway, NJ **2019** pp. 1–6.
[19] P. Kanerva, *Cogn. Comput.* **2009**, *1*, 139.
[20] A. Rahimi, et al., in *ISLPED*, ACM, New York, NY **2016**, pp. 64–69.
[21] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, T. Rosing, in *2019 IEEE 12th International Conf. on Cloud Computing (CLOUD)*, IEEE, Piscataway, NJ **2019** pp. 435–446.
[22] M. Nazemi, A. Esmaili, A. Fayyazi, M. Pedram, in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, Piscataway, NJ **2020**, pp. 1–9.
[23] A. Mitrokhin, P. Sutor, C. Fermüller, Y. Aloimonos, *Sci. Robot.* **2019**, *4*, 30.
[24] E. P. Frady, S. J. Kent, B. A. Olshausen, F. T. Sommer, *Neural Comput.* **2020**, *32*, 2311.
[25] E. P. Frady, D. Kleyko, F. T. Sommer, *Neural Comput.* **2018**, *30*, 1449.
[26] Y. Ni, H. Chen, P. Poduval, Z. Zou, P. Mercati, M. Imani, in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, IEEE, Piscataway, NJ **2023**, pp. 01–09.
[27] P. Poduval, M. Issa, F. Imani, C. Zhuo, X. Yin, H. Najafi, M. Imani, in *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, IEEE, Piscataway, NJ **2021**, pp. 1–6.
[28] M. Imani, A. Zakeri, H. Chen, T. Kim, P. Poduval, H. Lee, Y. Kim, E. Sadredini, F. Imani, in *Proc. of the 59th ACM/IEEE Design Automation Conf.*, IEEE, Piscataway, NJ **2022**, pp. 31–36.
[29] A. Cano, N. Matsumoto, E. Ping, M. Imani, in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, Piscataway, NJ **2021**.
[30] D. Kleyko, M. Davies, E. P. Frady, P. Kanerva, S. J. Kent, B. A. Olshausen, E. Osipov, J. M. Rabaey, D. A. Rachkovskij, A. Rahimi, F. T. Sommer, arXiv:2106.05268, **2021**.
[31] P. Poduval, Z. Zou, H. Najafi, H. Homayoun, M. Imani, in *2021 58th ACM/IEEE Design Automation Conf. (DAC)*, IEEE, Piscataway, NJ **2021**, pp. 1195–1200.
[32] P. Poduval, Y. Ni, Y. Kim, K. Ni, R. Kumar, R. Cammarota, M. Imani, in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, Piscataway, NJ **2021**.
[33] M. Imani, A. Rahimi, D. Kong, T. Rosing, J. M. Rabaey, in *2017 IEEE Int. Symp. on High Performance Computer Architecture (HPCA)*, IEEE, Piscataway, NJ **2017**, pp. 445–456.
[34] A. Moin, A. Zhou, A. Rahimi, A. Menon, S. Benatti, G. Alexandrov, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt, L. Benini, A. C. Arias, J. M. Rabaey, *Nat. Electron.* **2021**, *4*, 54.
[35] G. Karunaratne, M. Schmuck, M. Le Gallo, G. Cherubini, L. Benini, A. Sebastian, A. Rahimi, *Nat. Commun.* **2021**, *12*, 2468.
[36] P. Poduval, Y. Ni, Y. Kim, K. Ni, R. Kumar, R. Cammarota, M. Imani, in *Proc. of the 59th ACM/IEEE Design Automation Conf.*, IEEE, Piscataway, NJ **2022**, pp. 367–372.
[37] B. Khaleghi, M. Imani, T. Rosing, arXiv:2005.06716, **2020**.
[38] P. Kanerva, in *ICANN 98*, Springer, New York, NY **1998**, pp. 387–392.
[39] M. Imani, D. Kong, A. Rahimi, T. Rosing, in *2017 IEEE International Conf. on Rebooting Computing (ICRC)*, IEEE, Piscataway, NJ **2017** pp. 1–8.
[40] P. Neubert, S. Schubert, P. Protzel, *KI* **2019**, *33*, 319.
[41] C. Yeung, P. Poduval, M. Imani, *arXiv:2403.13218*, **2024**.
[42] Y. Kim, M. Imani, N. Moshiri, T. Rosing, in *2020 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, IEEE, Piscataway, NJ **2020**, pp. 115–120.
[43] Z. Zou, H. Chen, P. Poduval, Y. Kim, M. Imani, E. Sadredini, R. Cammarota, M. Imani, in *Proc. of the 49th Annual International Symp. on Computer Architecture*, Association for Computing Machinery, New York, NY, United States **2022**, pp. 656–669.

[44] P. Poduval, Z. Zou, X. Yin, E. Sadredini, M. Imani, in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, IEEE, Piscataway, NJ **2021**, pp. 781–786.
[45] P. Poduval, H. Alimohamadi, A. Zakeri, F. Imani, M. H. Najafi, T. Givargis, M. Imani, *Front. Neurosci.* **2022**, *16* 757125.
[46] K. Pagiamtzis, A. Sheikholeslami, *IEEE J. Solid-State Circuits* **2006**, *41*, 712.
[47] X. Yin, K. Ni, D. Reis, S. Datta, M. Niemier, X. S. Hu, *IEEE Trans. Circuits Syst. II: Express Briefs* **2018**, *66*, 1577.
[48] S. Salahuddin, K. Ni, S. Datta, *Nat. Electron.* **2018**, *1*, 442.
[49] T. Mikolajick, U. Schroeder, S. Slesazeck, *IEEE Trans. Electron Dev.* **2020**, *67*, 1434.
[50] T. Feist, *White Paper* **2012**, *5*.
[51] K. Ni, M. Jerry, J. A. Smith, S. Datta, in *2018 IEEE Symposium on VLSI Technology*, IEEE, Piscataway, NJ **2018**, pp. 131–132.
[52] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, J. Kopena, *SIGCOMM Demonstr.* **2008**, *14*, 527.
[53] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, S. Ghemawat, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, et al., *arXiv:1603.04467*, **2016**.
[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, É. Duchesnay, *J. Mach. Learn. Res.* **2011**, *12*, 2825.
[55] X. Shen, W. Liu, I. Tsang, F. Shen, Q.-S. Sun, in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. *31*, AAAI Press, Washington, DC, USA **2017**.
[56] S. Jockel, *Crossmodal learning and prediction of autobiographical episodic experiences using a sparse distributed memory* **2010**.
[57] P. Kanerva, J. Kristofersson, A. Holst, in *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, Vol. *1036*. Citeseer **2000**.
[58] A. Joshi, J. Halseth, P. Kanerva, *Quantum Interaction 2016 Conf. Proceedings*, in Press.
[59] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, J. M. Rabaeyet, in *ICRC*, IEEE, Piscataway, NJ **2016**, pp. 1–8.
[60] A. Burrello, K. Schindler, L. Benini, A. Rahimi, in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, IEEE, Piscataway, NJ **2018**, pp. 1–4.
[61] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, J. M. Rabaey, *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *99*, 1.
[62] D. Kleyko, E. Osipov, in *2014 International Conference on Computer and Information Sciences (ICCOINS)*, IEEE, Piscataway, NJ **2014**, pp. 1–6.
[63] D. Kleyko, E. Osipov, N. Papakonstantinou, V. Vyatkin, *IEEE Access* **2018**, *6* 30766.
[64] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, F. T. Sommer, *arXiv:2109.03429*, **2021**.
[65] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, T. Rosing, in *Proceedings of the 56th Annual Design Automation Conf. 2019*, IEEE Press **2019**, pp. 1–6.
[66] Pecan Street Dataport, https://dataport.cloud/ (accessed: December 2022).
[67] A. Reiss, D. Stricker, in ISWC, IEEE, Piscataway, NJ **2012**, pp. 108–109.
[68] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, *Commun. ACM* **2016**, *59*, 56.
[69] Y. Kim, P. Mercati, A. More, E. Shriver, T. Rosing, in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, Piscataway, NJ **2017**, pp. 683–690.
[70] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Proc. IEEE* **1998**, *86*, 2278.
[71] D. Anguita, et al., in *AAL* Springer, New York, NY **2012** pp. 216–223.