Advancing Hyperdimensional Computing Based on Trainable Encoding and Adaptive Training for Efficient and Accurate Learning

JISEUNG KIM, Department of Electrical Engineering and Computer Science, DGIST, Daegu, Korea (the Republic of)
HYUNSEI LEE, Department of Electrical Engineering and Computer Science, DGIST, Daegu, Korea (the Republic of)
MOHSEN IMANI, UC Irvine, Irvine, United States
YESEONG KIM, DGIST, Daegu, Korea (the Republic of)

Hyperdimensional computing (HDC) is a computing paradigm inspired by the mechanisms of human memory, characterizing data through high-dimensional vector representations, known as hypervectors. Recent advancements in HDC have explored its potential as a learning model, leveraging its straightforward arithmetic and high efficiency. The traditional HDC frameworks are hampered by two primary static elements: randomly generated encoders and fixed learning rates. These static components significantly limit model adaptability and accuracy. The static, randomly generated encoders, while ensuring high-dimensional representation, fail to adapt to evolving data relationships, thereby constraining the model's ability to accurately capture and learn from complex patterns. Similarly, the fixed nature of the learning rate does not account for the varying needs of the training process over time, hindering efficient convergence and optimal performance. This paper introduces TrainableHD, a novel HDC framework that enables dynamic training of the randomly generated encoder depending on the feedback of the learning data, thereby addressing the static nature of conventional HDC encoders. TrainableHD also enhances the training performance by incorporating adaptive optimizer algorithms in learning the hypervectors. We further refine TrainableHD with effective quantization to enhance efficiency, allowing the execution of the inference phase in low-precision accelerators. Our evaluations demonstrate that TrainableHD significantly improves HDC accuracy by up to 27.99% (averaging 7.02%) without additional computational costs during inference, achieving a performance level comparable to state-of-the-art deep learning models. Furthermore, TrainableHD is optimized for execution speed and energy efficiency. Compared to deep learning on a low-power GPU platform like NVIDIA Jetson Xavier, TrainableHD is 56.4 times faster and 73 times more energy efficient. This efficiency is further augmented through the use of Encoder Interval Training (EIT) and adaptive optimizer algorithms, enhancing the training process without compromising the model's accuracy.

CCS Concepts: • **Theory of computation → Random projections and metric embeddings**; • **Computing methodologies → Machine learning algorithms**.

1 Introduction

In today's computing landscape, we are witnessing an unprecedented surge in the volume of data being generated and processed. This exponential increase, emanating from a diverse array of sources and applications, poses significant challenges but also opens new avenues for the field of computational science. Traditional data

Authors' Contact Information: Jiseung Kim, Department of Electrical Engineering and Computer Science, DGIST, Daegu, Korea (the Republic of); e-mail: js980408@dgist.ac.kr; Hyunsei Lee, Department of Electrical Engineering and Computer Science, DGIST, Daegu, Korea (the Republic of); e-mail: wwhslee@dgist.ac.kr; Mohsen Imani, UC Irvine, Irvine, California, United States; e-mail: m.imani@uci.edu; Yeseong Kim, DGIST, Daegu, Korea (the Republic of); e-mail: yeseongkim@dgist.ac.kr.

ACM 1557-7309/2024/6-ART
https://doi.org/10.1145/3665891

processing methods, despite their robustness, are finding it increasingly difficult to cope with this burgeoning tide of data.

In response to these challenges, state-of-the-art solutions often turn to machine learning, with Neural Networks (NNs) emerging as a prominent approach. NNs are lauded for their exceptional capacity to learn from and make informed decisions based on extensive datasets. However, their reliance on substantial computational power often necessitates the use of advanced systems or servers. This requirement renders NNs less practical for a wide range of smaller, resource-constrained devices that are frequently at the forefront of data generation. This mismatch between the computational demands of NNs and computing capabilities has steered researchers towards alternative learning methodologies.

Hyperdimensional computing (HDC), inspired by the complex processes of the human brain, represents a novel computing paradigm that revolutionizes traditional data representation and processing methods. Utilizing high-dimensional vectors, termed hypervectors, HDC enables parallel computations and noise-tolerant learning, distinguishing it from conventional systems that rely on specific positions of elements, like 32/64-bit words [10].

The essence of HDC lies in its unique approach to data representation and processing. By *encoding* raw input data into hypervectors and integrating them with HDC operations, the system mimics cognitive functions such as memorization and information association. Then, during the subsequent *model training*, HDC culminates in the development of new hypervectors that represent different data classes, with inference conducted by measuring hyperspace distances between an input's hypervector encoding and each class hypervector [21].

For over a decade, numerous endeavors have sought to enhance the learning efficacy of HDC-based methodologies. Despite these efforts, conventional HDC-based learning systems continue to exhibit relatively low accuracy levels when compared with contemporary state-of-the-art learning mechanisms like deep learning. This discrepancy predominantly stems from the *static* nature that pervades the learning process within HDC frameworks for both the encoding phase and the model training phases. Each phase encounters its distinct challenge, constraining the adaptability and overall efficacy of HDC systems.

First, the encoding phase, which is responsible for transforming raw data into high-dimensional representations, is fundamental to achieving superior learning quality as it lays the groundwork for all subsequent learning processes. Traditional methods employ a high-dimensional matrix of base hypervectors, which are randomly generated before training [2, 36]. However, this approach also presents a significant limitation: once generated, these encoders remain static for the most previous HDC studies [2, 7, 10, 13, 16, 18, 20, 21, 23, 26, 34, 36, 46] The static nature of the encoding procedure prevents the system from dynamically adapting to evolving data feature relationships, compelling the use of excessively large dimensions (e.g., $D = 10,000$) to preserve accuracy. Such an approach inevitably leads to learning inefficiencies and scalability issues.

The challenges extend into the model training phase, where the primary issue lies in the static nature of learning rate adjustments. The introduction of an adaptive learning rate scheduler represents an important advancement in addressing this limitation. Unlike static learning rates, which remain constant throughout the training, we could employ an adaptive learning rate scheduler to dynamically adjust the rate in response to the training's progression. While there have been attempts to incorporate such adaptive mechanisms in HDC, these efforts often rely on predefined scheduling policies, such as user-defined linear adjustments [12], which lack the flexibility and responsiveness of truly adaptive systems. This discrepancy with modern learning algorithms like deep learning hampers HDC models' convergence and performance. This dynamic adjustment capability is crucial for modernizing HDC systems to achieve higher learning quality akin to deep learning.

Addressing these challenges due to the static learning nature existing in both the encoding and model training phases, we introduce the TrainableHD framework. Our approach is characterized by two contributions that collectively enable dynamic and adaptive learning to modernize the HDC paradigm: the implementation of a dynamic encoder and the integration of adaptive optimizer algorithms, all while incorporating an efficient quantization method.

The first major feature of the TrainableHD framework is its dynamic encoder. Unlike traditional HDC methods that utilize static, randomly generated encoders, TrainableHD's encoder is designed to adapt dynamically. It allows for continuous updates to the base hypervectors during the training process. This dynamic adaptation significantly enhances the model's accuracy and efficiency by enabling a more nuanced understanding of data feature relationships, as evidenced by related research [16, 18]. Importantly, TrainableHD achieves these enhancements without introducing additional operations during inference, maintaining high performance and efficiency.

Further extending the capabilities of TrainableHD, we leverage state-of-the-art adaptive optimizer algorithms used in deep learning, such as Adam and Adagrad, to fine-tune the learning rate and the magnitude of hypervector updates. This second innovation ensures that the training phase is optimized, facilitating faster convergence and greater overall efficiency [23]. By integrating these adaptive optimizer algorithms, TrainableHD not only surpasses traditional HDC methods in terms of adaptability and performance but also aligns HDC practices with the sophistication observed in modern learning algorithms like deep learning.

The contributions of this paper are summarized as follows:

(1) **Introduction of** TrainableHD**:** We propose TrainableHD, a novel HDC framework that enables the learning of a novel HDC encoder, dynamically updating it during training.
(2) **Integration of Adaptive Optimizer:** We extend TrainableHD by incorporating adaptive optimizer algorithms that dynamically adjust the learning rate and updated hypervectors to optimize training efficiency and accuracy.
(3) **Optimization of Encoder Training:** We present an optimization technique to reduce the overhead of encoder training, ensuring that the encoder is updated only when necessary for improved performance.
(4) **Efficient Inference through Quantization:** TrainableHD supports quantization-aware training for HDC learning to facilitate efficient inference across diverse platforms, including CPUs with SIMD, Tensor Cores (GPU), and FPGA, while maintaining comparable accuracy.

Our evaluation demonstrates that TrainableHD, enhanced with the adaptive optimizer algorithms, not only surpasses the accuracy of existing HDC learning methods by up to 27.99% (average 7.02%) but also achieves this without additional computational costs during inference. Furthermore, combined with quantization, TrainableHD shows a significant increase in speed and energy efficiency, being 56.4× faster and 73× more energy-efficient compared to deep learning on platforms like the NVIDIA Jetson Xavier.

2 Preliminary and Related Work

2.1 Hyperdimensional Computing

HDC is a paradigm where data is represented using high-dimensional vectors, known as hypervectors [15]. This section introduces foundational concepts of HDC and its learning mechanisms, which are essential for understanding our proposed method, TrainableHD.

2.1.1 Holistic Representation HDC utilizes hypervectors, analogous to how the human brain employs neurons and synapses to process stimuli. These hypervectors are high-dimensional, typically containing thousands of dimensions, and distribute information equally across all components. Unlike traditional computing methods where significance is often tied to specific bit indices of elements, as in 32/64-bit words, HDC ensures a more holistic representation of data. To determine the similarity between two entities represented as hypervectors, the dot product or cosine similarity is commonly used.[1]

The high dimensionality of these vectors (e.g., $D > 1000$) allows for quasi-orthogonality, meaning two randomly generated hypervectors are likely to be near orthogonal, a crucial feature for HDC's noise and error tolerance.

[1]In this work, we use the dot product, denoted as $\delta(\cdot)$, for the similarity measure.

For example, two bipolar hypervectors randomly generated will be near orthogonal if they have a zero dot product, $\langle \mathbf{X}, \mathbf{Y} \rangle \approx 0$, where dimension D is sufficiently high ($D > 1000$).[2] This occurs because the two random hypervectors $\mathbf{X}, \mathbf{Y}$ are likely to have a $D/2$ overlap in bits with a standard deviation of $\sqrt{D/4}$. The independent and identically distributed (i.i.d.) components and high dimensions of hypervectors allow significant noise and errors to be added before they break orthogonality with other hypervectors. Additionally, the holistic nature of hypervectors makes them error-tolerant.

2.1.2 HDC Arithmetic In HDC, hypervectors undergo specific arithmetic operations to achieve various cognitive functionalities. For example, the *bundling* operation, denoted by $\oplus$, combines hypervectors through bit-wise addition, resulting in a composite hypervector reflective of its constituents and effectively mimicking the memorization of different information. Meanwhile, *binding*, represented by $\otimes$, is a bit-wise matrix multiplication operation that creates a new hypervector encapsulating the association of its factors, useful for associating two different concepts and creating another near-orthogonal hypervector representing a new concept different from their origins.

With these operations, diverse HDC-based learning tasks can be implemented where data points in hyperspace are represented by hypervectors. For instance, through a series of binding or bundling operations applied to related hypervectors, class hypervectors can be learned, signifying representative high-dimensional value patterns in their classification category. During inference, the similarity between query hypervectors and the encoded, trained class hypervectors is used to determine classification.

2.2 Existing Encoding Methods in HDC

Encoding significantly influences the accuracy and complexity of HDC learning models [23]. It involves mapping original data points into hyperspace, ensuring proximity for similar vectors and pseudo-orthogonality for unrelated ones. We briefly examine three state-of-the-art encoding methods: *ID-level encoding* [11], *random projection encoding* [10], and *nonlinear encoding* [13].

ID-level encoding, first introduced for learning in speech recognition tasks, accounts for a data sample's feature values and their index with two different hypervectors: *Level hypervectors* ($\mathbf{L}$), which quantize input feature values, and *ID hypervectors* ($\mathbf{I}$), randomly generated to represent unique positions of input features. The Level and ID hypervectors for each feature value in a dataset sample are bound together (i.e., $\mathbf{L_f} \otimes \mathbf{I_f}$ where f is the feature index), and the resulting bound hypervectors corresponding to each feature are bundled together ($\sum \mathbf{L_f} \otimes \mathbf{I_f}$) to form an encoded hypervector.

Random projection encoding addresses the low accuracy of binary-based ID-level encoding by utilizing floating-point operations during encoding. It generates F random bipolar base hypervectors of D dimensions, where F is the number of features in a dataset sample. Each feature of a data sample is bound (i.e., $\otimes$) with the randomly generated base hypervector matrix to produce the encoded hypervector by binding per-feature hypervectors.

Nonlinear encoding, one of the most advanced encodings, samples elements of base hypervectors from a Gaussian distribution, creating floating-point hypervectors similarly to random projection, but also applies nonlinear functions, such as the cosine function and/or sign function, to binarize hypervectors and amplify nonlinearity in their element distributions.

To train HDC learning models, the transformation of data samples into hypervectors constitutes a critical phase, exerting a substantial influence on both the accuracy and complexity of the model. Existing HDC learning methods [2, 7, 10, 13, 16, 18, 20, 21, 23, 26, 34, 36, 46] still use the static encoders, potentially overlooking valuable insights that can be gleaned from training samples. To mitigate this issue, the work presented in [47] introduced an alternative method, ManiHD, which incorporates manifold projection prior to the static HDC encoder. However,

[2]In this work, we use Latin capitals for hypervectors.

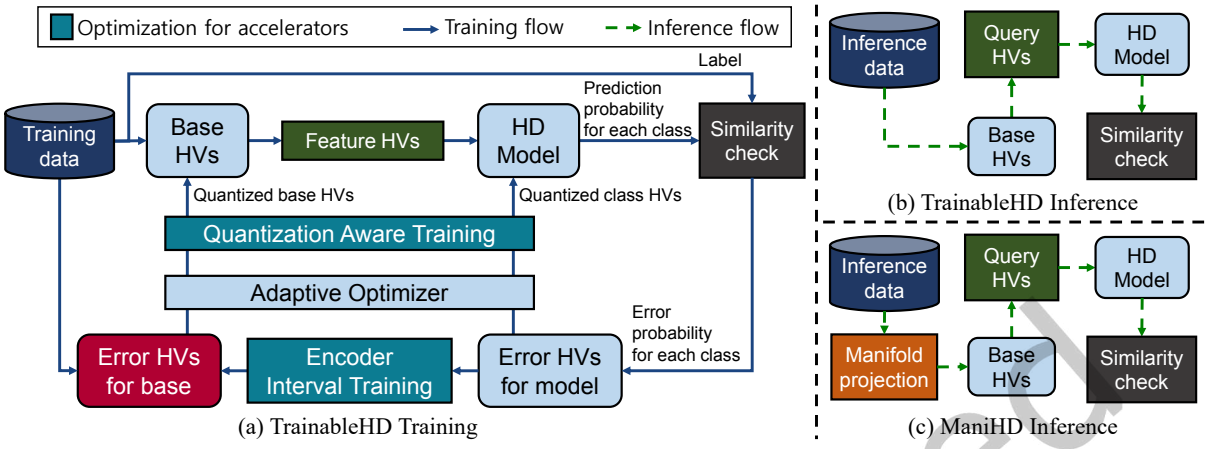


Fig. 1. An Overview of TrainableHD (a) Training, (b) Inference, and (c) ManiHD [47] Inference

ManiHD does not differentiate between class-specific characteristics and continues to rely on a static encoder, which can lead to suboptimal accuracy levels. Our analysis across various datasets revealed that the manifold projection process in ManiHD requires approximately 2.95 times longer for encoding, compared to other methods.

Addressing this challenge, this paper extends our prior study [17] to enable dynamic HDC encoder training along with adaptive learning optimizers for high accuracy and efficiency. We focus on how to explicitly learn the encoder itself by extracting and utilizing knowledge from training samples without introducing additional overheads during the inference process. In particular, we adopt nonlinear encoding as our baseline, given its demonstrated superiority in accuracy over other techniques. However, unlike the original nonlinear encoding method outlined in [13], our strategy actively seeks to identify the most suitable, representative high-dimensional patterns for each raw feature.

3 TrainableHD Overview

Figure 1 provides an illustrative overview of the TrainableHD learning framework. Our training methodology learns the following two hypervector representations: (i) the *base hypervectors* of the encoder, initially generated with random components from a Gaussian distribution, and (ii) the *class hypervectors*, which depict high-dimensional patterns for each class, starting with zero-value components.

In the training phase, TrainableHD first encodes the training samples into *feature hypervectors* using the current base hypervectors. The training process advances by evaluating the similarity between these feature hypervectors and each class hypervector. The HDC module classifies the training data based on the class hypervector exhibiting the highest similarity. Per-class errors, calculated using similarity values and ground-truth labels, guide the update of class hypervectors, enhancing prediction accuracy in future iterations.

The innovation of TrainableHD lies in its encoder training technique, which efficiently translates per-class scalar errors into per-feature hypervector errors, thus updating the base hypervectors. This process is detailed in Section 5. TrainableHD also incorporates adaptive optimizer algorithms in the HDC contexts (Section 5.1), traditionally used in deep learning, to optimize training performance. These algorithms, particularly valuable given the added complexity of updating base hypervectors, significantly enhance both accuracy and efficiency.

We also introduce two optimization strategies in TrainableHD to improve the efficiency of acceleration platforms. The first one is Encoder Interval Training (EIT) discussed in Section 6.1, an optimization technique that dictates encoder training frequency based on necessity. This adaptive approach ensures that training occurs

efficiently with a mini-batch over multiple epochs, effectively converging the accuracy of the model. Additionally, we designed a method for training the model with low precision hypervector elements, such as 8-bit integers, employing Quantization-Aware Training (QAT) detailed in Section 6.2. This approach enables the creation of efficient, quantized models without compromising on the model's inferential accuracy.

During the inference phase, fully accelerated by quantization, the trained encoder generates the *query hypervector* from inference data, incurring no extra computational costs compared to existing HDC learning solutions. The HDC model then performs similarity computation between the query hypervector and the class hypervectors to determine the class of the sample (Figure 1b). As highlighted, ManiHD [47] (Figure 1c), which uses manifold projection to mitigate the issue of static HDC encoders, incurs additional computational costs; TrainableHD does not add any computational overhead during the encoding process even in the inference phase.

4 HDC Learning with Dynamic Encoder Training

4.1 Encoding Principle

Much like how the human brain with millions of neurons and synapses activates upon input stimuli, HDC leverages high-dimensional space to represent*any entities*. The high-dimensional vectors, or hypervectors, encapsulating a holistic representation [15], distribute information uniformly across all components, ensuring a comprehensive and integrated data representation. As discussed in Section 2.2, in HDC, the generation of base hypervectors, which form the foundation of this encoding process, is typically achieved through random sampling from bipolar values {-1,1} [10] or a Gaussian distribution $N(\mu, \sigma^2)$ [13] to attain higher accuracy.

However, a limitation arises in traditional HDC methods as they do not alter the base hypervectors post-creation, leading to potential inaccuracies in representing correlated features. We here discuss the key properties of the general encoding procedure exploited in our dynamic encoding approach.

Consider a vector of scalars, $\vec{v}(\in \mathbb{R}^p)$, represented as $\langle v_1, \cdots, v_p \rangle$, which needs to be encoded into hypervectors. The codebook $\mathbb{C}(\in \mathbb{R}^{p \times D}) = \langle \mathbf{C}_1, \cdots, \mathbf{C}_p \rangle$ contains D-dimensional hypervector representations for each element in $\vec{v}$. State-of-the-art encoding methods, including random projection [10] and non-linear encoding [13], can be expressed as a series of operations:

$$v_1 \otimes \mathbf{C}_1 \oplus v_2 \otimes \mathbf{C}_2 \oplus \cdots v_p \otimes \mathbf{C}_p$$

where $\otimes$ represents the binding operation, associating different information through element-wise multiplication, and $\oplus$ is the bundling operation, combining diverse information into a single hypervector through element-wise addition. Post these operations, an activation function such as $cos(\cdot)$ or $sign(\cdot)$ is usually applied.

We can conceptualize HDC encoding as an *interdimensional mapping* that transcends various domains represented by different vector bases. We define the interdimensional mapping function as $\mathbf{H} = \phi_{\mathbb{C}}^{p \rightarrow q}(\vec{v})$. This process maps information from a real coordinate space of p dimensions to another hyperspace, characterized by a set of hypervectors, $\mathbb{C}$. For example, the original static encoding can be interpreted as a function that transmits the information stored in $\vec{v}$ into a hypervector $\mathbf{H}(\in \mathbb{R}^q)$, by referencing codewords in $\mathbb{C}$.

In the TrainableHD framework, we apply this interdimensional mapping principle to encode the error values in scalar vectors (observed during the tradining) to the domain of class hypervectors through another encoding process, $\phi_{\mathbb{K}}^{k \rightarrow D}(\vec{v})$, using the class hypervectors' codebook, $\mathbb{K}$. This approach allows us to utilize the inherent properties of HDC encoding to effectively translate per-class scalar errors into modifications of the base hypervector, thereby enhancing accuracy and adaptability in HDC.

4.2 HDC Model Training

The training process of TrainableHD, detailed in Algorithm 1, commences with the encoding of the feature hypervector, $\mathbf{H}$, utilizing the $sign(\cdot)$ function for binarization (❹). It transforms the raw input data into a binary

hypervector format, which is foundational for the subsequent learning phases. The ensuing stages include updating (i) the class hypervectors, $\mathbb{K}$ (❷), and (ii) the base hypervectors, $\mathbb{B}$ (❸).

Several methodologies exist for learning the class hypervectors $\mathbb{K}$ in HDC [11, 35]; in our work, we leverage the state-of-the-art HDC learning process introduced in [21]. It refines $\mathbb{K}$ based on the amount of per-class scalar errors $\vec{e}$ associated with each current class hypervector. In this paper, we exploit the same information of the scalar errors $\vec{e}$ to update the base hypervectors by identifying the underlying reasons attributed to the errors. This approach in TrainableHD contrasts with previous HDC encoding techniques, which do not train the base hypervectors of the encoder, i.e., relying on static and randomly generated encoders. By dynamically updating both the class and base hypervectors during training, TrainableHD addresses the key limitations of static encoding—namely, the disregard for the relationships between input data features and the consequent necessity for large dimensions.

Algorithm 1 Training procedure of TrainableHD

for $\vec{f}$ in training datasets **do**
  // ❶ **Encoding**
  $\mathbf{X} \leftarrow \phi_{\mathbb{B}}^{f \rightarrow D}(\vec{f})$; $\mathbf{H} \leftarrow sign(\mathbf{X})$
  // ❷ **Updating class HVs**
  $\vec{s} \leftarrow softmax(\delta(\mathbf{H}, \mathbb{K}))$; $\vec{e} \leftarrow \vec{o} - \vec{s}$
  $\Theta \leftarrow \vec{e} \times \mathbf{H}$
  $\Theta' \leftarrow \lambda \times \Theta$
  $\mathbb{K} \leftarrow \mathbb{K} \oplus \Theta'$
  // ❸ **Updating base HVs**
  $\mathbf{F}_{bz} \leftarrow \mathbf{I} - tanh(\mathbf{X})^2$
  $\mathbf{F}_{err} \leftarrow \phi_{\mathbb{K}}^{k \rightarrow D}(\vec{e})$
  $\mathbf{E} \leftarrow \mathbf{F}_{err} \otimes \mathbf{F}_{bz}$
  $\mathbf{\Delta} \leftarrow \vec{f} \times \mathbf{E}$
  $\mathbf{\Delta}' \leftarrow \lambda \times \mathbf{\Delta}$
  $\mathbb{B} \leftarrow \mathbb{B} \oplus \mathbf{\Delta}'$
end for

Figure 2 delineates the process through which TrainableHD updates the base hypervectors. In TrainableHD, the training of base hypervectors, $\mathbb{B}$, which initially encode scalar features into hypervectors, are strategically directed to convert the magnitude of per-class errors into that of per-feature errors. TrainableHD accomplishes this through a two-step methodology: (i) encoding the errors experienced by individual training samples into a hypervector, called the *sample error hypervector* (**E**), and (ii) deducing the per-feature error hypervector from **E**.
Step 1: Encoding the sample-wise error in a hypervector: TrainableHD encodes the sample error hypervector, **E**, which includes the hypervector-type information of *how much error occurs for a single sample*. The encoded hypervector, **X**, eventually contributes to the scalar errors, $\vec{e}$, through two following computations: (i) the binarization, (i.e., due to $\mathbf{H} \leftarrow sign(\mathbf{X})$ in ❶ of Algorithm 1) and (ii) the discrepancy with the class hypervectors, (i.e., due to $\delta(\mathbf{H}, \mathbb{K})$ in ❷). TrainableHD represents the two *factors* in a form of hypervectors, $\mathbf{F}_{bz}$ and $\mathbf{F}_{err}$. Figure 2❷ illustrates how we compute each factor.

• (❷-a) The binarization factor, F*bz*, is calculated using $\mathbf{I} - tanh(\mathbf{X})^2$, where **I** is a unit hypervector and $tanh(\cdot)$ represents the hyperbolic tangent function. Since the binarization function, $sign(\cdot)$, amplifies the hypervector element of **X** in the range of $[-1, +1]$, an element value closer to 0 may create higher errors in the prediction,

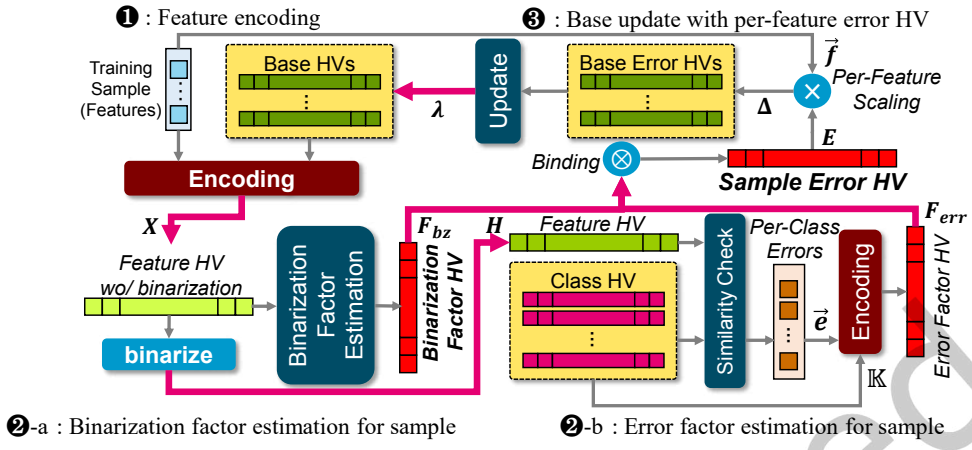


Fig. 2. An Illustration of Base Hypervector Training

where the impact of each element on the error is bound within the same range. We exploit the square of tanh(·), which is suitable to explain these impacts of the hypervector elements on the errors.

• (❷-b) Next, we compute the second factor due to the class hypervectors, $\mathbf{F}_{err}$. As discussed in Section 4.1, we can treat the HDC encoding as an interdimensional mapping. Thus, we encode the per-class error of $\vec{e}$ by using the class hypervectors $\mathbb{K}$ as the basis of the hyperspace of the encoded samples by $\mathbf{F}_{err} = \phi_{\mathbb{K}}^{k \to D}(\vec{e})$, meaning that $\mathbf{F}_{err}$ *bundles all per-class error hypervectors* scaled with the corresponding error value.

TrainableHD then synthesizes the sample-wise error hypervector $\mathbf{E}$ by binding these two encoded factors through $\mathbf{E} \leftarrow \mathbf{F}_{err} \otimes \mathbf{F}_{bz}$.

Step 2: Estimating the per-feature errors in hypervectors: In this phase (❸), TrainableHD produces *base error hypervectors*, referred to as $\mathbf{\Delta}$. These hypervectors estimate errors at the feature level arising from individual base hypervectors. The sample error hypervector, representing the errors for each sample, is then utilized to extrapolate these errors across the feature domain. This extrapolation, guided by the assumption that higher feature values in the raw training samples correspond to more significant errors, is expressed as $\mathbf{\Delta} = \vec{f} \times \mathbf{E}$. Consequently, $\mathbf{\Delta}$ comprises f hypervectors, each encapsulating the quantum of per-feature error. The final update of the base hypervectors is then achieved by bundling these base error hypervectors, moderated by a learning rate.

5 Dynamic HDC Training with Learning Rate Optimization

5.1 Integration of Learning Rate Optimization in HDC

In the field of HDC, traditional algorithms have generally bypassed the complexity of optimizing learning rates, as noted in previous studies [12]. Such algorithms, only updating class hypervectors, have demonstrated relatively quick convergence, especially when compared to the more intricate processes involved in deep learning. However, TrainableHD represents a shift from these conventional HDC approaches. By introducing dynamic updates not just to class hypervectors but also to base hypervectors, TrainableHD navigates a more expansive training landscape. The base hypervectors, due to their broader scope — potentially encompassing a greater number of features than the number of classes — suggest that adopting a more detailed, deep learning-like approach to update the hypervectors with finer-grained scales could be advantageous. This refined strategy in TrainableHD aims to cultivate high-quality models while maintaining expedient convergence rates.

Table 1. Modification on Algorithm 1 to apply optimizer algorithms

Algorithm	Hypervector Update	Notations
SGD (Alg. 1)	$\omega'_t \leftarrow \lambda \times \omega_t$	λ: Learning rate ω_t: Error hypervectors (Θ, $\mathbf{\Delta}$) ω'_t: Scaled error hypervectors (Θ', $\mathbf{\Delta'}$) ω'_{t-1}: Scaled error hypervectors on previous epoch Ω: Accumulated hypervector-type errors ϵ: Small term to avoid division in 0 β_1, β_2, γ: Decay rate η: Momentum term
Adagrad	$\omega_t \leftarrow \Omega + \omega_t^2$ $\omega'_t \leftarrow \frac{\lambda}{\sqrt{\Omega+\epsilon}} \times \omega_t$	
AdaDelta	$\omega_t \leftarrow \gamma \times \Omega + (1-\gamma) \times \omega_t^2$ $\omega'_t \leftarrow \lambda \times \frac{\sqrt{\omega'_{t-1}}}{\sqrt{\Omega+\epsilon}} \times \omega_t$	
RMSProp	$\omega_t \leftarrow \gamma \times \Omega + (1-\gamma) \times \omega_t^2$ $\omega'_t \leftarrow \frac{\lambda}{\sqrt{\Omega+\epsilon}} \times \omega_t$	
Momentum	$\omega'_t \leftarrow \lambda \times \omega_t - \eta \times \omega'_{t-1}$	
NAG	$\omega'_t \leftarrow (1-\eta) \times \lambda \times \omega_t - \eta \times \omega'_{t-1}$	
Adam	$m_t = \beta_1 m_{t-1} + (1-\beta_1)\omega_t$ $v_t = \beta_2 v_{t-1} + (1-\beta_2)\omega_t^2$ $\omega'_t \leftarrow \lambda \times \frac{m_t}{1-\beta_1^t} \div \sqrt{\frac{v_t}{1-\beta_2^t} + \epsilon}$	

Incorporating state-of-the-art learning rate optimization algorithms into the HDC framework necessitates a systematic modification for algorithmic integration. The TrainableHD framework identifies two primary steps for the hypervector update process: the class hypervectors (Θ') and the base hypervectors (Δ'), both modulated by a learning rate (λ). In TrainableHD's training framework, as detailed in Algorithm 1, two steps for the two hypervector updates are defined: the updating of class hypervectors (Θ', Line 7) and base hypervectors (Δ', Line 14) with λ. For these update processes, we can adopt well-known learning rate optimization techniques from the deep learning domain, such as Adagrad [4] and Adam [22], into our HDC training algorithm.

For example, applying the Adagrad algorithm to class hypervector updates involves modifying the procedure to calculate the optimal update state: $\Theta_{acc} \leftarrow \Theta_{acc} + \Theta^2$; $\mathbb{K} \leftarrow \mathbb{K} \oplus (\frac{\lambda}{\sqrt{\Theta_{acc}+\epsilon}} \times \Theta)$. In this formula, Θ_{acc} denotes the accumulated squared errors for class hypervectors, and ϵ is a small term added to avoid division by zero. We can similarly apply the principle of learning rate adaptation to the process of updating the base hypervectors. This integration allows HDC to enhance the training process for both class and base hypervectors, ultimately contributing to the achievement of high accuracy in the model. This strategy is detailed in Table 1, highlighting the modifications aligned with various optimization algorithms to foster a dynamic and efficient training process within the HDC paradigm.

5.2 Implication of different optimization algorithms on HDC

The selection and integration of optimizer algorithms play a crucial role in enhancing the training efficiency and overall performance of models. The TrainableHD framework incorporates a range of optimization algorithms; each of these optimizers brings distinct characteristics and potential drawbacks to the HDC training process, In this section, we present a comprehensive discussion on their expected behavior and compatibility with the HDC learning.

SGD (Stochastic Gradient Descent) [40] SGD, known for its simplicity and widespread use in traditional HDC studies, employs a fixed learning rate, providing a stable but potentially rigid learning process. While SGD has demonstrated rapid convergence and comparable accuracy in many HDC applications, its fixed learning rate can be a significant drawback in scenarios where data distributions evolve over time, limiting the model's ability to adapt to new information effectively.

Adagrad [4] It addresses the limitation of SGD by adapting the learning rate to each parameter, allowing for more sophisticated updates based on the frequency of feature occurrence. In particular, it excels in environments

with holistically represented data due to its unique mechanism of adapting the learning rate to the frequency of features. This characteristic is particularly beneficial in HDC, where encoding often results in high-dimensional vectors with significant holographic representation. The accumulator in Adagrad, which scales the learning rate inversely proportional to the square root of the sum of all squared updates, ensures that infrequent but informative features receive larger updates. This approach can significantly enhance learning in the early stages by allowing rapid adaptation to the most informative features of the data.

AdaDelta [43] It designed to overcome the diminishing learning rate issue of Adagrad, focuses on using a more recent update accumulation window to adjust the learning rates. While this approach mitigates some of Adagrad's limitations, it could lead to a decline in performance in HDC's context, which often benefits from rapid learning capabilities rather than prolonged iterative adjustments.

RMSProp [42] Similar to AdaDelta, it modifies Adagrad's approach by employing a moving average to determine the learning rate, thereby ensuring that the learning rate does not decrease too quickly. This optimizer is better suited for non-stationary problems and can offer improvements over AdaDelta in HDC by maintaining an effective pace of learning.

Momentum [37] and NAG (Nesterov Accelerated Gradient) [33] They introduce velocity components to the optimization process, enabling the model to navigate the optimization landscape more smoothly and potentially escape local minima more effectively. While these methods can accelerate convergence in HDC models by incorporating past updates into the current update, they might also introduce oscillations during the training process, particularly due to the high-dimensional spaces characteristic of HDC. This oscillations and subsequent computation overheads would result in negative impacts on the learning performance, as the HDC usually aims a fast learner as compared to other state-of-the-art learning algorithms like deep learning. Additionally, these optimizer algorithms are known to be sensitive to hyperparameters such as the momentum coefficient and the learning rate. This sensitivity can complicate the process of training the optimal model, often requiring numerous iterations to identify the most effective parameter settings.

Adam [22] Combining the strengths of adaptive learning rates with momentum, it adjusts the learning rate based on both the first and second moments of the updates makes it robust across a wide range of HDC tasks. However, it would be slower at initial training epochs, which is potential drawbacks in the domain of HDC. Also, its complexity and the need for careful hyperparameter tuning can be viewed as potential drawbacks, especially in scenarios where computational resources are limited.

While each optimizer brings unique advantages to the HDC training process, their effectiveness can be contingent upon the specific characteristics of the HDC task, including the stage of learning and the computational constraints. The subsequent sections, especially Section 7.2, discuss the profound impacts of these optimization techniques on the performance of TrainableHD to examine the potential of the integration for thje learning rate optimization in the HDC domain.

6 Optimization Strategies for Acceleration

6.1 Encoder Interval Training

In TrainableHD's training methodology, continuously encoding feature hypervectors with updated base hypervectors for each iteration can significantly increase the complexity of HDC-based learning. To mitigate this issue, we have implemented an optimization technique known as *Encoder Interval Training* (EIT). This technique is specifically designed to streamline the training process of TrainableHD by reducing the frequency of encoding operations.

As illustrated in Figure 3, EIT operates by storing the values of feature hypervectors in memory after their initial encoding. In the initial iteration, TrainableHD conducts the standard procedure of encoding feature hypervectors. Subsequently, for the following (n-1) iterations, where n represents a hyperparameter that defines the EIT cycle,

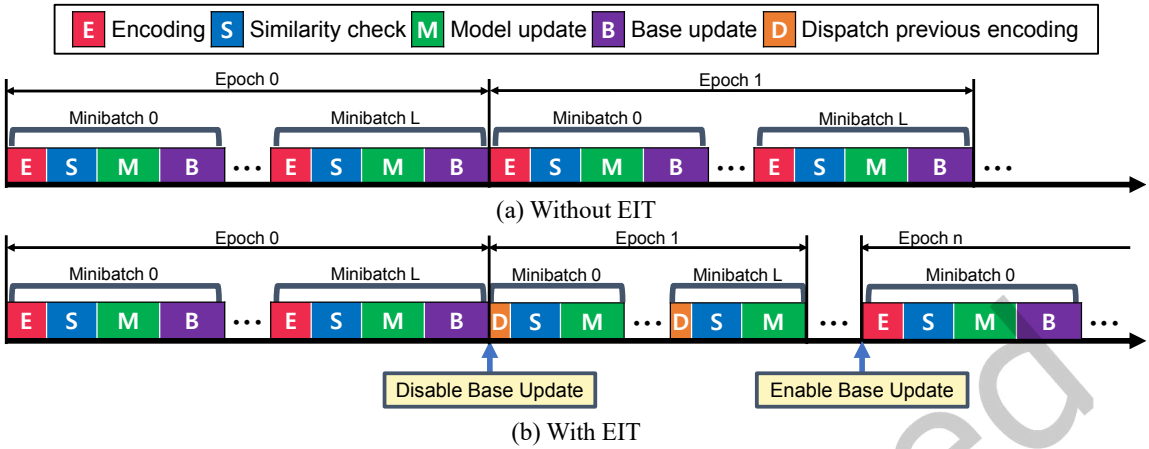


Fig. 3. An example of Encoder Interval Training

the training process utilizes these previously stored feature hypervectors, bypassing the need for re-encoding at each step.

It is important to note that while the reuse of feature hypervectors reduces the frequency of encoding, the base hypervectors continue to undergo updates during these iterations. This ongoing refinement of base hypervectors ensures that even with the reuse of feature hypervectors, the overall encoding quality remains high. The inherent noise tolerance of HDC minimizes potential loss in learning quality, ensuring efficient updates of feature hypervectors only when significant changes are observed.

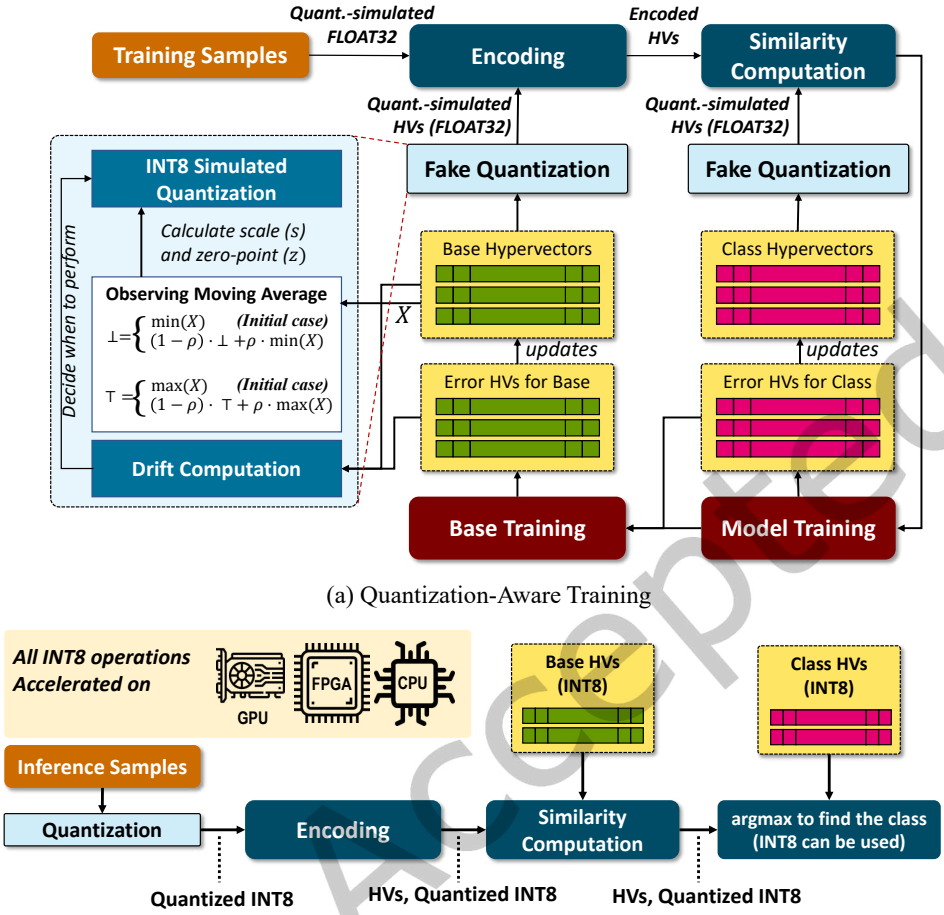
This cyclical process of reusing and updating feature hypervectors continues throughout the training, with a re-encoding operation performed at every n^{th} iteration. This structured approach of EIT not only optimizes the encoding process within TrainableHD but also contributes to an overall more efficient and effective training regimen. The EIT cycle is repeated until the training reaches its done, ensuring a balance between computational efficiency and the quality of learning.

6.2 Acceleration with Quantization

In TrainableHD, we adopt a quantization strategy to address the computational challenges associated with floating-point operations. Contemporary HDC algorithms, notably those based on non-linear encoding [21], typically operate with encoded binary hypervectors to achieve high accuracy. Yet, their internal mechanisms, including encoding and similarity search procedures essential for class computation using learned class hypervectors, also rely on floating-point computations. This reliance becomes a significant challenge when seeking efficient acceleration, the primary issue that TrainableHD addresses through the application of quantization.

Quantization, widely used in deep learning for efficient inference, has been adapted in TrainableHD to address the specific needs of HDC. In TrainableHD, quantization is utilized to streamline computations while ensuring minimal impact on accuracy by leveraging the inherent robustness of HDC. The versatility of quantization is further exemplified in TrainableHD by the variety of methodologies it accommodates. These methods range from static uniform or logarithmic quantization to post-training quantization with dynamic range adjustments and mixed-precision quantization approaches.

Specifically, TrainableHD employs an advanced Quantization-Aware Training (QAT) method [5, 24, 32, 45] that adeptly transitions hypervectors from floating-point formats to quantized representations during training.



(a) Quantization-Aware Training

(b) Inference with Quantization

Fig. 4. TrainableHD Quantization

That is, during the training process, it emulates the exact quantization procedure in software on GPGPU (or CPU) platforms to train the quantized model, ensuring that the transition to the quantized states incurs minimal accuracy loss. Upon completion of the training phase, the quantized model is primed for deployment on acceleration platforms, e.g., CUDA Tensor Cores or FPGAs, which support quantized operations. This deployment performs inference only with quantized representations, offering both high accuracy and efficiency.

While INT8 quantization is our primary focus in this paper, given its prevalence in modern hardware accelerators like NVIDIA Tensor Cores, TrainableHD's quantization framework is flexible enough to extend to other precision levels with minimal adjustments. This adaptability is critical, allowing TrainableHD to be effectively deployed across various hardware platforms, thereby enhancing its utility and efficiency in diverse hardware platforms that support quantization.

6.2.1 Training Quantized Hypervectors The adaptation of QAT in TrainableHD, as illustrated in Figure 4a, enables the modern quantization approaches to traditional HDC training methodologies. During training, TrainableHD

subjects both the input features and the current base hypervectors to a quantization simulation process termed *fake quantization*. This process calculates their elements to 8-bit integers (INT8) (i.e., by clamping and rounding hypervector elements to produce the version of the inputs in 8-bit integers) while retaining their storage in the floating-point (FLOAT32) format. The purpose of this approach is to model the effects of quantization on the hypervectors without bearing the extensive computational costs typically associated with actual INT8 quantization during iterative training updates. Class hypervectors are similarly processed through fake quantization.

Upon completing the training phase, we convert the trained base and class hypervectors into INT8 representations. This step allows the entire inference process to be conducted within the INT8 domain, effectively reducing computational overhead, as depicted in Figure 4b.

The challenge in training with floating-point quantization lies in accurately setting the quantization parameters. To tackle this, TrainableHD employs an *affine transformation* method, quantifying each input x using a scale factor s and a zero-mapping value z:

$$quant(x) = min(max(round(s \cdot x + z), 2^{b-1} - 1), -2^{b-1}))$$

Here, $b = 8$ for INT8 quantization. This method dynamically adapts to changing bounds of the input x, learning the parameters s and z throughout the training iterations. We track the moving averages of the minimum ($\perp$) and maximum ($\top$) values of the hypervectors to be quantized (Figure 4a), with a decay rate empirically set at $\rho = 0.01$, thereby evenly dividing the range $[\perp, \top]$ into quantized points. Upon the completion of the QAT process, the final quantized model is derived using the learned scale factors and zero points, enabling it to perform on inference platforms through actual hardware quantization equivalently.

6.2.2 QAT Performance Optimization Our approach to Quantization-Aware Training (QAT) is tailored to enhance inference efficiency, though it does introduce additional training overhead from the fake quantization process. Given that HDC is discussed as a candidate for online learning solutions in resource-constrained environments [8–10, 28–31, 44, 47], these added costs can become burdensome in certain deployment scenarios. To counteract the additional overhead predominantly caused by fake quantization, we have developed a targeted optimization technique named Drift-Aware Update (DAU). DAU is designed to strategically determine the most opportune moments for implementing fake quantization, thereby optimizing the training process.

Let us recall Algorithm 1, which outlines the training methodology of TrainableHD. We adapt the base and class hypervectors in response to hypervector-type errors (computed through the chosen learning optimizer), namely $\mathbf{\Theta}'$ and $\mathbf{\Delta}'$ as discussed in Section 4.2. Given HDC's inherent tolerance to noise, minor discrepancies in the base and class hypervectors are unlikely to substantially impact the overall training results, allowing us the discretion to occasionally bypass the fake quantization step.

Consequently, DAU initiates fake quantization solely when it detects significant accumulations of changes, termed as *drift*, in $\mathbf{\Theta}'$ and $\mathbf{\Delta}'$. More formally, DAU triggers the fake quantization process for class hypervectors if the sum of changes, $\sum_i |\mathbf{\Theta}'_i|/|\mathbb{K}|$, exceeds a set threshold ϵ. Similarly, for base hypervectors, fake quantization occurs when $\sum_j |\mathbf{\Delta}'_j|/|\mathbb{B}| > \epsilon$. This threshold is empirically set at 0.01 (1%) in our implementation, a conservative measure to ensure that QAT operates accurately and effectively across the majority of the training iterations. This optimization not only enhances the efficiency of the training process but also aligns with the dynamic requirements of various HDC applications.

7 Experimental Results

7.1 Experimental Setup

We have implemented the training procedure of the TrainableHD framework using PyTorch running on NVIDIA GeForce RTX 3090. The inference procedure was implemented on various acceleration platforms that support both floating-point and integer vector operations, including CPU (Intel Xeon Silver 4110), low-power GPU

(Nvidia Jetson Xavier), and FPGA (Xilinx Zynq-7000). To evaluate the performance, we employed specific tools for each platform: Intel RAPL was used for measuring execution time and power consumption on the CPU, Nvidia Nsight was utilized for the GPU assessments, and the Xilinx Vitis toolkit was applied for evaluating the FPGA. These tools provided comprehensive metrics to analyze the efficiency and effectiveness of the TrainableHD framework across different hardware environments. Our training framework is open-sourced and available at https://github.com/CELL-DGIST/HDZoo/trainableHD. This repository provides access to the resources necessary for exploring and implementing our method, as well as benchmarking it with other representative HDC-based learning strategies.

7.1.1 Implementation Methodologies for Inference Tasks For the CPU-based acceleration of inference, we leveraged Facebook's FBGEMM library, a state-of-the-art solution known for supporting optimized INT8 operations. This library utilizes x86 SIMD instructions and multithreading capabilities. We further enhanced the FBGEMM library to include support for the sign function. Given that the base and class hypervectors remain constant post-deployment, once trained, we pre-arranged the storage order of hypermatrix elements. This pre-arrangement ensured a fully sequential memory access pattern during General Matrix Multiply (GEMM) operations, optimizing performance.

In the case of GPU acceleration, our focus was on exploiting the Tensor Cores within NVIDIA's Jetson Xavier. We extended XCelHD [16], the CUDA implementation of HDC, to facilitate the quantized execution of the trainableHD framework. This extension involved mapping HDC operations to CUDA's cuBLAS APIs. Additionally, we developed an in-place element update function for intermediate results, allowing the support of the sign function without encountering uncoalesced memory accesses.

For FPGA implementation, our proposed quantization method was designed to be both highly accurate and efficient, circumventing the need for resource-intensive DSP units as seen in previous works [10]. Utilizing the Xilinx Vitis framework, we implemented GEMM and reduction operations on a systolic array structure primarily relying on Look-Up Tables (LUTs) for computation. This approach was further refined by pre-loading invariant base and class hypervectors into the systolic array's buffer, facilitating their reuse for multiple inputs without necessitating additional host communications. This proactive loading strategy significantly enhances the efficiency of our FPGA-based inference processes.

7.1.2 Baselines and Datasets For our comparative analysis, TrainableHD is benchmarked against three distinct models: (i) a state-of-the-art HDC learning method (Baseline) that employs a static nonlinear encoder for retraining class hypervectors [13], (ii) ManiHD (ManiHD), which integrates manifold projection into HDC to overcome the issue of the static HDC encoder [47], and (iii) deep learning models (DNN) that have been optimized using Ray Tune for maximal accuracy. These deep learning models were configured with varying hyperparameters, including batch sizes up to 64, architecture depths of 5 layers, and up to 512 neurons, trained over 50 epochs.

To conduct a fair evaluation, TrainableHD, Baseline, and ManiHD were each retrained for 50 epochs. We empirically selected the learning rate (λ) in the range $[10^{-5}, 10^{-2}]$ with common hyperparameter search. The datasets used for this evaluation, detailed in Table 2, encompass a broad spectrum of practical applications. These range from IoT and edge systems [1, 3, 38, 39], such as ECG-based emotion detection and human activity recognition, to image-based classification tasks typically used for benchmarking HDC learning [6, 19, 25, 27] like face detection and character recognition.

7.2 Classification Accuracy

Figure 5 illustrates the accuracy comparison among various learning methods. For both Baseline and TrainableHD, we conducted measurements using two different hypervector dimensions: 3K and 10K. The results demonstrated that TrainableHD consistently achieves higher accuracy levels than Baseline, a state-of-the-art method employing

Table 2. Evalution Datasets

Name	Description	N_{train}	N_{test}	k	f
EMOTION [1]	Emotion recognition from ECG signal	1705	427	3	1500
FACEA [19]	Face recognition	22441	2494	2	512
FACE [19]	Face recognition	22441	2494	2	608
HACT [3]	Human activity recognition	7352	2947	6	1152
HEART [14]	MIT-BIH Arrhythmia dataset	119560	4000	5	187
ISOLET [3]	Voice recognition	6238	1559	26	617
MAR [27]	Plant classification	1440	160	100	64
MNIST [25]	Hand-written digit classification	60000	10000	10	784
PAMAP2 [38]	Physical activity monitoring dataset	16384	16384	5	27
SA12 [39]	Smartphone-based activity recognition	6213	1554	12	561
TEX [27]	Plant classification	1439	160	100	64
UCIHAR [3]	Human activity recognition	7352	2947	6	561

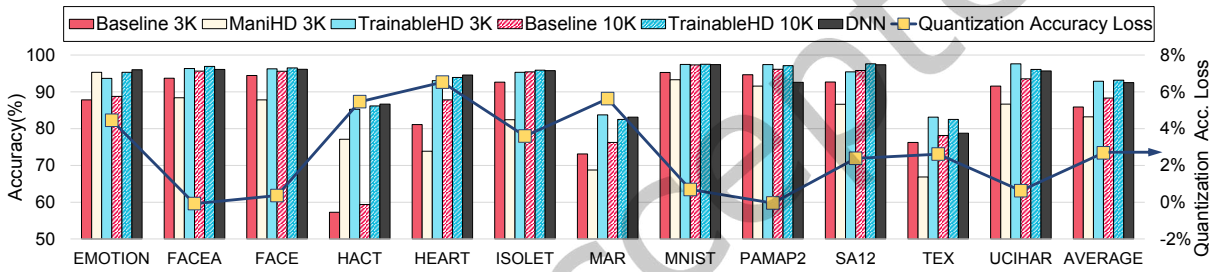


Fig. 5. Accuracy Comparison

a nonlinear encoder. For instance, in the case of the HACT dataset, Baseline recorded classification accuracies of 57.28% and 59.35% for dimensions $D = 3,000$ and $D = 10,000$, respectively. In contrast, TrainableHD achieved significantly higher accuracies of 85.27% for $D = 3,000$ and 86.16% for $D = 10,000$. We observed that Quantization-Aware Training (QAT) introduces only a relatively low accuracy loss, e.g., 0.69% for MNIST dataset, 2.68% on average.

On average, when comparing the accuracies at the same hypervector dimensions, TrainableHD outperformed Baseline by 7.02% and 4.86% for $D = 3,000$ and $D = 10,000$, respectively. These results are on par with state-of-the-art deep learning models that have been fine-tuned for optimal accuracy. In comparisons with ManiHD at $D = 3,000$, TrainableHD surpassed ManiHD in all but one dataset, averaging an accuracy improvement of 9.68%. It is important to note, however, that ManiHD utilizes manifold projection, which introduces significant overhead during inference.

To delve deeper into why TrainableHD surpasses other HDC learning methods, we analyzed the changes in training and testing accuracy across epochs. A notable characteristic of HDC learning is its ability to learn a high-quality model within a relatively small number of epochs. As demonstrated in Figure 6, both Baseline and TrainableHD reached high levels of training accuracy in the initial epochs. While Baseline showed comparable training accuracy (i.e., for the training dataset) to TrainableHD, TrainableHD exhibits higher testing accuracy over the training epochs. This was attributed to the tendency of the baseline HDC-based learning to overfit the training datasets. TrainableHD, on the other hand, avoided such overfitting in the early stages of training, leading

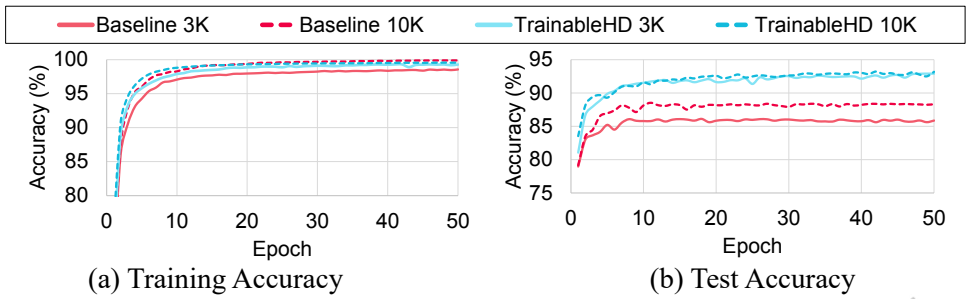


Fig. 6. Learning Accuracy Changes over Epochs

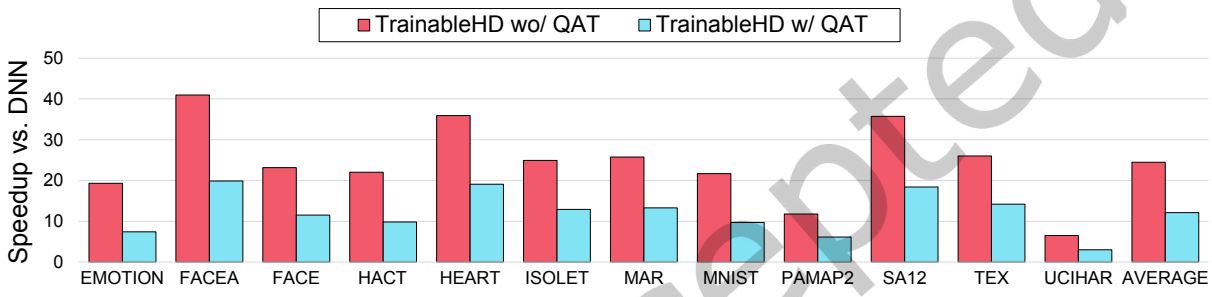


Fig. 7. Training Speedup of TrainableHD over DNN

to superior testing accuracy — a more accurate indicator of real-world prediction performance for robust online learning.

7.3 Efficiency Evaluation

Training Efficiency The training efficiency of TrainableHD was assessed in comparison to traditional Deep Neural Networks (DNNs). In Figure 7, we present the results from two versions of TrainableHD: one trained without Quantization-Aware Training (QAT) and the other with QAT, set against a standard DNN model. Remarkably, TrainableHD without quantization shows an average training performance improvement of 24.48 times. Incorporating quantization, which enables the generation of INT8-quantized models, further enhances efficiency during inference deployment. Despite additional computational demands introduced by the fake quantization simulation, TrainableHD with QAT still achieves a 12.13 times speedup compared to DNNs.

Inference Efficiency Figure 8 provides a comparative analysis[3] of speedup and energy efficiency improvements of TrainableHD over DNN inference on the GPU. For this evaluation, a dimension of $D = 3,000$ was used for TrainableHD, which still outperforms the learning quality of Baseline method with $D = 10,000$. TrainableHD shows significantly higher learning efficiency compared to DNNs. For instance, on the same GPU platform, TrainableHD, even without utilizing quantization, is 56.4 times faster and 73 times more energy-efficient. When quantization is enabled, TrainableHD's performance improves by an additional 3.1 times without any loss in accuracy.

[3]We do not compare with the computational costs of the Baseline since it is exactly the same with the TrainableHD, as no extra computational procedures are added during the inference.

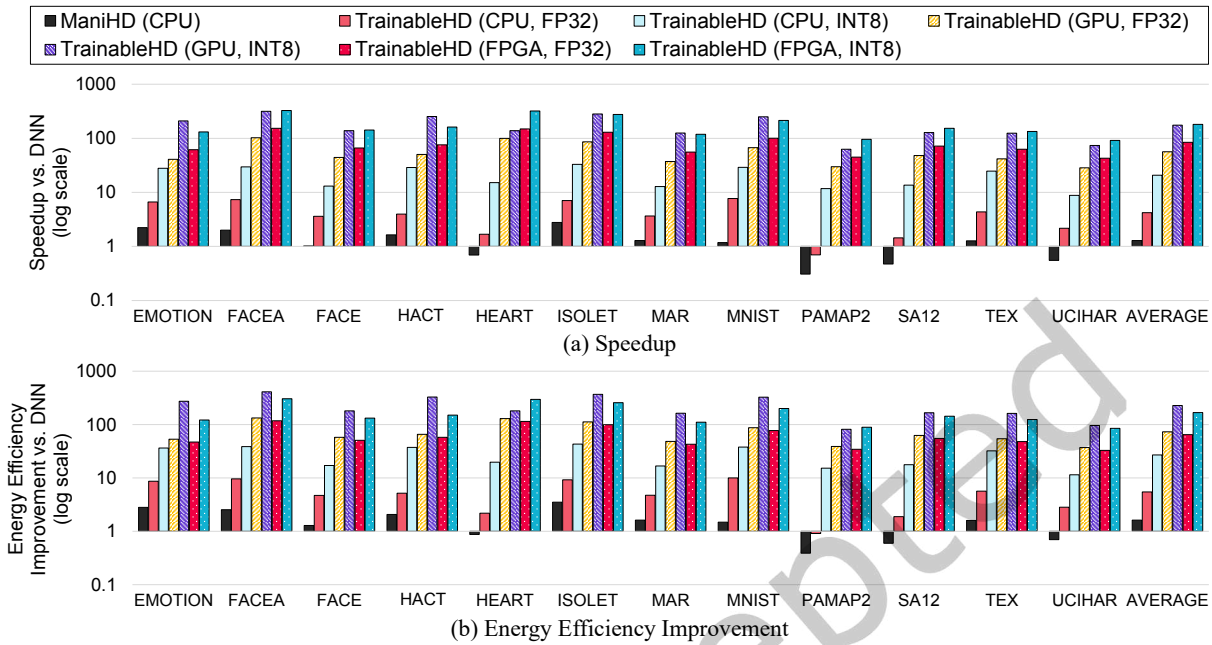


Fig. 8. Efficiency Comparison for different learning algorithms and platform combinations

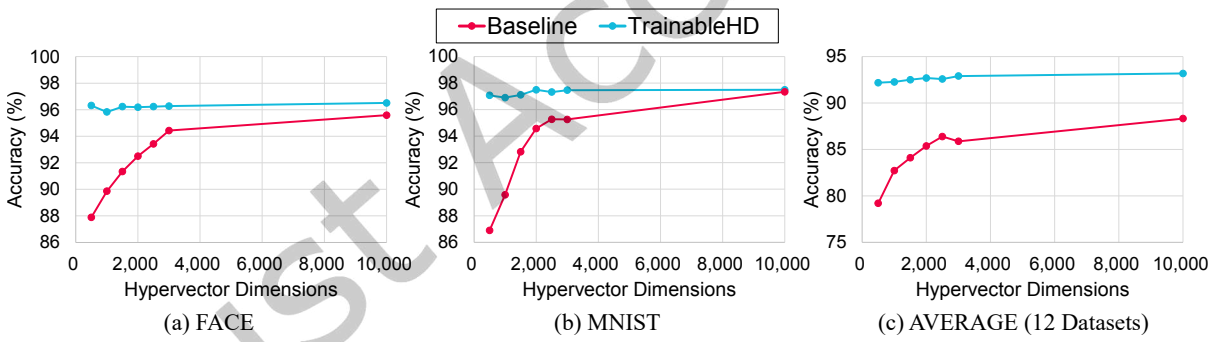


Fig. 9. Impact of Dimension Reduction

It's also noteworthy that TrainableHD's lightweight framework enables high efficiency on CPUs, beneficial for edge or cloud computing scenarios where on-device accelerators might not be available. With quantization, TrainableHD offers a performance that is 20.7 times faster than DNNs on GPU. Additionally, when implemented on the FPGA unit, TrainableHD can provide an impressive 180.8 times speedup and 167.8 times better energy efficiency compared to DNNs.

TrainableHD with FP32 (and INT8) quantization demonstrates a 3.3 (and 16.1) times improvement over ManiHD. This enhancement is particularly significant considering that ManiHD incurs notable overheads to achieve higher accuracy, primarily due to the preprocessing CPU overhead of the manifold projection.

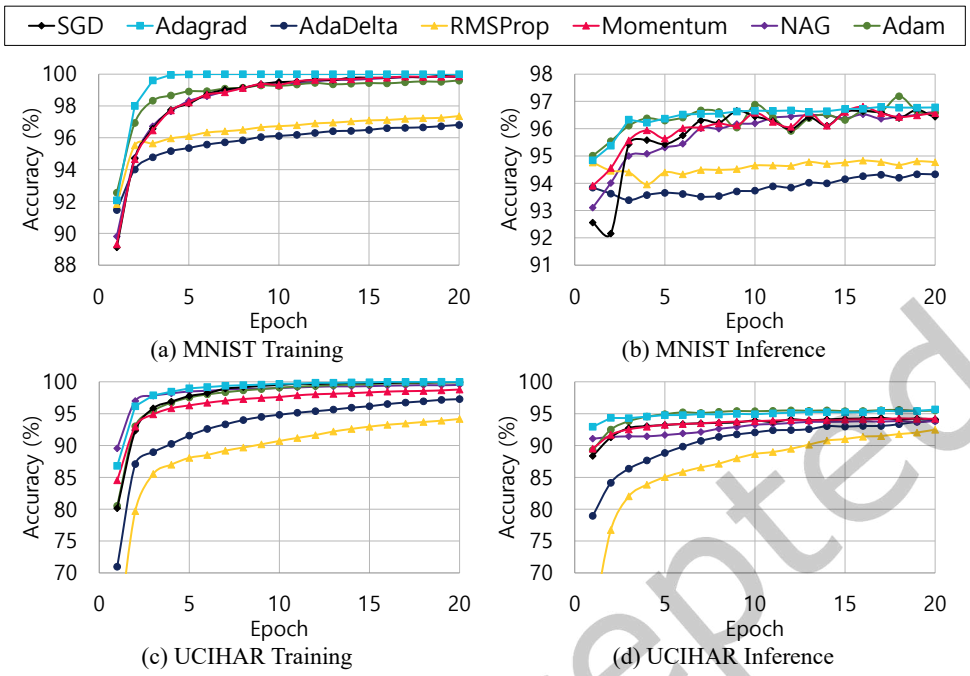


Fig. 10. Learning Accuracy Changes via Optimizer Algorithms

7.4 Dimension Reduction

Figure 9 presents a comparison of inference accuracies for TrainableHD and Baseline across varying hypervector dimensions. One of the key aspects of HDC is the potential for increased efficiency through dimension reduction, although this often comes at the expense of reduced accuracy. A critical factor in HDC is maintaining an adequate dimension size to accurately represent distances between the model and encoded hypervectors.

Even though the general trend of lower dimensions leads to a decline in accuracy and the minimal accuracy variations over dimensions exists due to inherent stochastic elements akin to deep learning, TrainableHD demonstrates a notable robustness to dimension reduction while still ensuring high accuracy. For instance, in MNIST dataset, Baseline achieves 97.34% test accuracy at $D = 10,000$, which is closely matched by TrainableHD's 96.90% accuracy at a significantly reduced dimension of $D = 1,000$. On average, the accuracy loss on $D = 500$ compared to $D = 10,000$ is only 0.99% for TrainableHD. This finding is remarkable as it suggests that TrainableHD can effectively operate with reduced dimensions without substantial accuracy loss. Moreover, TrainableHD with a dimension of $D = 500$ surpasses the accuracy of BaselineHD even at $D = 10,000$. This comparison underscores TrainableHD's capability to maintain high accuracy levels, even with a considerable reduction in hypervector dimensions.

7.5 Optimizer Algorithm Evaluation

In assessing the effectiveness of various optimizer algorithms on TrainableHD, we implemented a range of algorithms including Adagrad [4], Adam [22], RMSProp [42], Momentum [37], NAG [33], and SGD [40].[4] Figure 10

[4]Note that SGD, which employs a fixed learning rate for updating hypervectors, is the most widely used algorithm in HDC studies if they introduced a learning rate, including our previous work [17].

Table 3. Inference Accuracy Improvement Compared to SGD [17]

Dataset	Epoch 5						Epoch 50					
	Adagrad	AdaDelta	RMSProp	Momentum	NAG	Adam	Adagrad	AdaDelta	RMSProp	Momentum	NAG	Adam
EMOTION	0.47%	1.64%	-12.41%	-1.17%	**2.81%**	1.87%	-1.17%	-1.64%	-1.41%	-0.47%	-1.87%	-0.23%
FACEA	**1.12%**	0.32%	0.52%	-8.30%	0.12%	0.64%	0.64%	-0.16%	0.08%	0.04%	0.00%	**1.08%**
FACE	2.81%	1.92%	**2.85%**	2.37%	2.33%	1.60%	0.72%	0.12%	**1.36%**	0.36%	-0.20%	0.84%
HACT	-3.12%	-2.21%	-7.43%	-3.02%	-2.27%	-1.29%	1.90%	**3.36%**	3.29%	-1.70%	-1.87%	**3.36%**
HEART	0.12%	-7.15%	**1.90%**	-9.03%	-2.55%	-0.15%	2.18%	-5.72%	2.25%	1.73%	1.18%	**3.28%**
ISOLET	**2.69%**	-5.64%	-1.09%	0.71%	1.48%	1.60%	**1.03%**	-1.92%	0.71%	0.13%	0.13%	0.64%
MAR	2.50%	-21.25%	-8.75%	**4.38%**	**4.38%**	0.00%	0.00%	-0.62%	**1.25%**	-1.25%	-1.25%	**1.25%**
MNIST	**0.94%**	-1.78%	-0.52%	0.21%	-0.12%	0.87%	0.00%	-2.35%	**1.13%**	0.04%	0.03%	0.68%
PAMAP2	**0.54%**	**0.54%**	-3.33%	0.45%	0.45%	-0.07%	0.16%	-0.07%	0.16%	-0.01%	-0.01%	**0.18%**
SA12	1.67	-6.05%	-0.26%	1.48%	**1.93%**	0.13%	-0.13%	-2.25%	0.00%	-0.06%	0.06%	**0.58%**
TEX	**0.00%**	-3.75%	-6.25%	-29.38%	-1.88%	-2.50%	0.62%	**1.25%**	**1.25%**	**1.25%**	0.00%	0.00%
UCIHAR	1.56%	-4.38%	-8.18%	0.03%	-2.38%	**1.70%**	**1.39%**	0.64%	-0.17%	0.07%	0.51%	1.09%
AVERAGE	**0.94%**	-3.98%	-3.58%	-3.44%	0.36%	0.37%	0.61%	-0.78%	0.83%	0.01%	-0.27%	**1.06%**

illustrates the training and inference accuracy achieved across epochs with these optimizers. Our observation indicates that the Adam algorithm yields the best overall performance for TrainableHD. Notably, compared to our earlier version in [17], the application of the Adam algorithm resulted in an average accuracy improvement of 1.06%. During training, the SGD optimizer conventionally used in the HDC domain demonstrated comparable accuracy and rapid convergence with other sophisticated optimizers such as Adam and Adagrad. However, in terms of inference performance, the Adam algorithm achieved highly accurate results generally for most datasets in Table 3, consequently outperforming the traditional SGD approach.

Interestingly, the AdaDelta algorithm resulted in a notable decline in TrainableHD's performance across both the training and inference stages. This highlights the unique requirements for optimizers in HDC, contrasting with those in DNN. AdaDelta was originally designed to mitigate the diminishing learning rate issue of Adagrad [41], primarily by focusing on more recent training data for updates. Conversely, HDC is defined by its rapid learning capabilities, as discussed in 7.2, potentially obviating the need for prolonged iterative training. Thus, we anticipated that algorithms like Adagrad, which capitalize on insights for early training stages, would be more effective than AdaDelta in HDC contexts.

In alignment with our discussion in Section 5.2, Adagrad demonstrates competitive inference accuracy, particularly when the number of retraining sessions is limited. For example, as illustrated in Table 3, Adagrad outperforms other optimization algorithms in four out of twelve datasets during the initial stages of learning, e.g., at the fifth epoch. This observation also points out the importance of selecting an appropriate optimizer for specific datasets, with alternatives like RMSProp and NAG showing potential for superior accuracy under certain conditions. Such findings emphasize the advantage of leveraging HDC's rapid learning capabilities through strategic optimizer selection tailored to each dataset's unique characteristics. Nonetheless, extensive retraining over many epochs reveals that the Adam optimizer consistently delivers high performance, presenting its efficacy in enhancing the learning outcomes of HDC models over time.

We next examine the impact of the different optimization algorithms on the training execution time. As detailed in Table 4, we compared the training time over the 50 epoch retraining for various optimization algorithms against the baseline provided by SGD, which requires the least computation costs. Notably, applying the NAG algorithm was observed to extend the training time by approximately fivefold on average. This observation implies that we should consider the computational overhead introduced by some advanced optimizers to balance training efficiency alongside learning accuracy. For example, for HDC models that only can undergo fewer retraining cycles, the optimization algorithms such as Adagrad and RMSProp would be preferable choices as they offer a balanced improvement in learning quality without significantly impacting the execution time. This evaluation suggests a strategic selection of optimization algorithms based on the specific requirements of the

Table 4. Training Time Increment Compared to SGD [17]

Dataset	Adagrad	AdaDelta	RMSProp	Momentum	NAG	Adam
EMOTION	2.06×	2.82×	1.95×	7.49×	12.04×	2.50×
FACEA	1.17×	1.84×	1.29×	3.60×	6.29×	1.72×
FACE	1.25×	1.94×	1.43×	3.55×	6.17×	1.80×
HACT	1.62×	2.78×	1.91×	5.95×	10.36×	2.54×
HEART	1.24×	1.55×	1.32×	2.05×	3.02×	1.43×
ISOLET	1.12×	1.88×	1.38×	3.89×	6.84×	1.69×
MAR	1.21×	1.56×	1.27×	1.89×	2.71×	1.44×
MNIST	1.57×	2.51×	1.75×	4.75×	8.29×	2.29×
PAMAP2	1.24×	1.58×	1.32×	1.41×	1.72×	1.53×
SA12	1.17×	1.83×	1.32×	3.32×	5.58×	1.68×
TEX	1.20×	1.62×	1.30×	1.85×	2.74×	1.51×
UCIHAR	1.23×	1.92×	1.39×	3.45×	5.80×	1.77×
AVERAGE	1.34×	1.99×	1.47×	3.60×	5.96×	1.83×

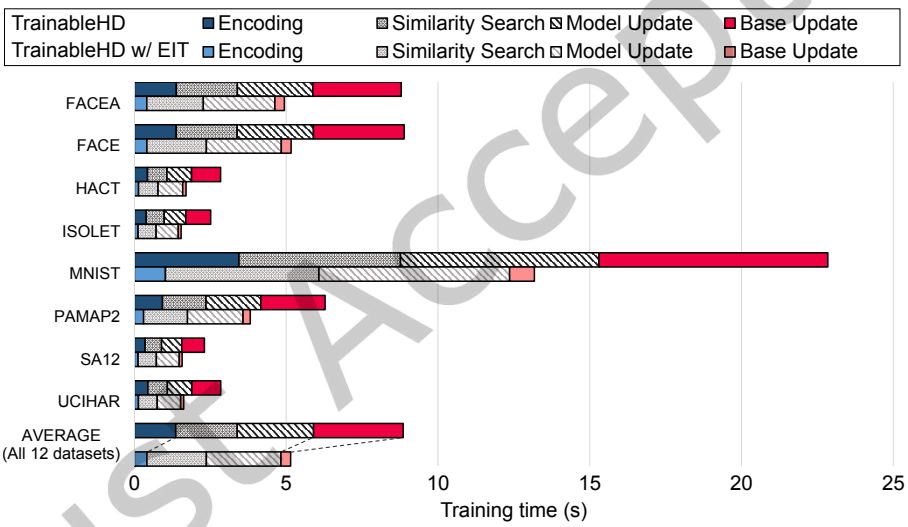


Fig. 11. Impact of Encoder Interval Training

training procedure and the computational constraints at hand, ensuring an optimal trade-off between model accuracy and training efficiency.

7.6 Impact of Acceleration Strategies

7.6.1 Encoder Interval Training (EIT) The EIT strategy, as previously outlined in Section 6.1, involves conducting encoder training at regular intervals rather than at every epoch. We evaluate this approach by measuring inference times with and without the use of EIT. Figure 11 illustrates a breakdown of training times for each scenario, using representative datasets. EIT has been found to substantially reduce the time required for both base hypervector updates (labeled 'Base Update') and the repeated encoding process (labeled 'Encoding'). Specifically,

Table 5. Accuracy Loss on Applying EIT

Dataset	Epoch 5							Epoch 50						
	SGD	Adagrad	AdaDelta	RMSProp	Mom.	NAG	Adam	SGD	Adagrad	AdaDelta	RMSProp	Mom.	NAG	Adam
EMOTION	12.65%	8.20%	2.11%	-13.82%	-3.98%	-0.47%	33.26%	0.23%	-0.23%	-0.23%	-2.11%	7.26%	2.58%	0.23%
FACEA	2.53%	0.40%	0.16%	0.40%	-7.14%	1.36%	-0.32%	0.80%	0.52%	-0.04%	-0.04%	-0.12%	-0.20%	0.92%
FACE	-2.57%	0.28%	0.04%	0.44%	0.92%	0.08%	-0.84%	-0.88%	0.00%	0.16%	0.80%	0.56%	0.04%	0.44%
HACT	5.26%	3.80%	-1.63%	-7.50%	-0.10%	-0.14%	1.97%	1.36%	1.93%	2.24%	3.02%	0.20%	0.27%	2.99%
HEART	16.20%	2.20%	-1.17%	0.68%	-6.63%	1.73%	2.85%	4.30%	2.15%	2.35%	1.08%	10.33%	1.78%	0.95%
ISOLET	0.19%	0.38%	1.28%	-1.41%	-0.64%	0.13%	1.09%	0.19%	0.13%	1.80%	0.38%	0.32%	0.38%	-0.26%
MAR	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-0.63%	-0.63%	1.25%	0.62%	0.00%	0.00%	1.88%
MNIST	1.72%	0.31%	0.81%	0.60%	0.90%	0.71%	0.85%	-0.15%	-0.01%	0.98%	1.46%	0.12%	0.41%	1.04%
PAMAP2	0.00%	-0.01%	0.98%	0.04%	0.00%	-0.01%	0.15%	-0.01%	-0.02%	0.02%	3.77%	0.01%	0.01%	0.05%
SA12	-0.06%	0.45%	0.06%	-1.09%	0.13%	0.32%	0.97%	0.32%	0.06%	0.39%	0.00%	0.39%	0.39%	0.64%
TEX	0.00%	0.63%	0.00%	-1.25%	0.00%	0.00%	-0.62%	0.00%	0.62%	0.00%	0.62%	17.50%	0.00%	0.00%
UCIHAR	-0.17%	0.64%	0.24%	0.03%	0.00%	-1.26%	0.71%	0.44%	0.68%	0.85%	0.71%	0.64%	4.68%	-0.14%
AVERAGE	2.98%	1.44%	0.24%	-1.91%	-1.38%	0.20%	3.34%	0.50%	0.43%	0.81%	0.86%	3.10%	0.86%	0.73%

our evaluations indicate that EIT decreases the time spent on base updates and encoding by 89.20% and 69.89%, respectively. This demonstrates the significant efficiency gains achieved through the EIT technique.

The EIT strategy is devised to enhance the efficiency of the training process by selectively bypassing the re-encoding of base hypervectors when non-critical. While this approach can significantly reduce the training time, it potentially compromises accuracy since it layers the exact updates of the base hypervector representations. In Table 5, we evaluate the impact of employing EIT on the accuracy of TrainableHD across different datasets, setting the cycle number of $n = 5$ for a direct comparison to the non-EIT scenario. For consistency, we applied the same learning rate for each combination of dataset and optimizer, which yielded the best inference accuracy without EIT.

Our findings indicate that while the EIT technique largely preserves a satisfactory balance between accuracy and efficiency in most evaluated cases, non-negligible accuracy degradation is observed for specific datasets when coupled with the Momentum optimizer. It is attributed to the sensitivity of the Momentum optimizer to the learning rate, as we discussed in Section 5.2, which would become critical under the EIT conditions. For example, using the standard learning rate setting for each dataset, the performance of the Momentum optimizer diminished, suggesting that its efficacy is contingent on optimal parameter configurations.

With further investigation into the role of the learning rate, the default settings optimal for most optimizers do not necessarily translate to the optimal settings for Momentum. We observed that adapting the learning rate for the Momentum optimizer significantly enhanced the accuracy. For instance, the TEX dataset experienced a reduction in accuracy loss from 17.50% to 5.00% upon adjusting the learning rate from $\lambda = 0.035$ to $\lambda = 0.001$. It presents that the adverse effects of EIT can be substantially mitigated through parameter optimizations.

On the other hand, our analysis also suggests that employing EIT with various optimizer algorithms generally incurs minimal accuracy losses, for example, an average loss of 0.50% for SGD, 0.43% for Adagrad and 0.73% for Adam. It indicates the potential for optimization algorithms to more effectively leverage EIT benefits through careful tuning of the configuration parameters, suggesting that they could harness the efficiency gains of EIT without significantly compromising model accuracy.

7.6.2 Drift-Aware Update (DAU) for QAT DAU, as detailed in Section 6.2, is another optimization technique implemented in our framework. It selectively applies simulated quantization during hypervector updates only when there are substantial changes, thus reducing unnecessary computational effort. To assess the effectiveness of DAU, we compared scenarios where DAU was either enabled or disabled, the latter involving continuous QAT and fake quantization at every update. As depicted in Figure 12, which examines ten representative datasets, DAU is shown to reduce QAT overhead by an average of 84.50%. This reduction is crucial for maintaining high

Fig. 12. Training Time Reduction of Drift-Aware Update

efficiency during training. Additionally, it is important to note that the use of DAU resulted in only minimal changes in accuracy, a benefit attributable to the holistic nature of the representation used in HDC.

8 Conclusion

This paper presents TrainableHD, an innovative framework that revitalizes HDC by introducing dynamic training capabilities for encoders and integrating adaptive optimizer algorithms. By moving beyond the constraints of static, randomly generated encoders typically used in HDC, TrainableHD adapts and evolves based on learning data feedback, marking a significant shift from traditional HDC methods. This dynamic approach to encoder training is a key factor in enhancing the accuracy and efficiency of the HDC model. The introduction of adaptive optimizer algorithms in the training process further refines TrainableHD, optimizing the training of hypervectors and contributing to the framework's overall efficiency. This enhancement ensures that TrainableHD not only maintains but also elevates the inherent advantages of HDC, such as straightforward arithmetic and high computational efficiency. Another advancement in TrainableHD is its implementation of effective quantization, which enables the execution of the inference phase on low-precision accelerators. This feature significantly boosts the framework's applicability in real-world scenarios, particularly on low-power platforms like NVIDIA Jetson Xavier, where TrainableHD outperforms state-of-the-art deep learning models in both speed and energy efficiency. Our comprehensive evaluations have demonstrated that TrainableHD achieves up to a 27.99% increase in accuracy (averaging 7.02%) compared to traditional HDC methods, without incurring additional computational costs. Moreover, TrainableHD's application of Encoder Interval Training (EIT) and adaptive optimizer algorithms further augments its performance, realizing a balance between training efficiency and model accuracy.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (No. 2018R1A5A1060031), Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT)

(No.2022-0-00991, 1T-1C DRAM Array Based High-Bandwidth, Ultra-High Efficiency Processing-in-Memory Accelerator) 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.

References

[1] Jordan J Bird, A Ekart, CD Buckingham, and Diego R Faria. 2019. Mental emotional sentiment classification with an eeg-based brain-machine interface. In *Proceedings of theInternational Conference on Digital Image and Signal Processing (DISP'19)*.

[2] Sohum Datta, Ryan A. G. Antonio, Aldrin R. S. Ison, and Jan M. Rabaey. 2019. A Programmable Hyper-Dimensional Processor Architecture for Human-Centric IoT. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 3 (2019), 439–452. https://doi.org/10.1109/JETCAS.2019.2935464

[3] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[4] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, 7 (2011).

[5] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. 2020. Learned Step Size Quantization. arXiv:1902.08153 [cs.LG]

[6] Michael Fink and Pietro Perona. 2022. Caltech 10k Web Faces. https://doi.org/10.22002/D1.20132

[7] Lulu Ge and Keshab K. Parhi. 2021. Seizure Detection Using Power Spectral Density via Hyperdimensional Computing. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 7858–7862. https://doi.org/10.1109/ICASSP39728.2021.9414083

[8] Alejandro Hernández-Cano, Namiko Matsumoto, Eric Ping, and Mohsen Imani. 2021. Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 56–61.

[9] Michael Hersche, Edoardo Mello Rella, Alfio Di Mauro, Luca Benini, and Abbas Rahimi. 2020. Integrating event-based dynamic vision sensors with sparse hyperdimensional computing: a low-power accelerator with online learning capability. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design* (Boston, Massachusetts) *(ISLPED '20)*. Association for Computing Machinery, New York, NY, USA, 169–174. https://doi.org/10.1145/3370748.3406560

[10] Mohsen Imani, Yeseong Kim, Sadegh Riazi, John Messerly, Patric Liu, Farinaz Koushanfar, and Tajana Rosing. 2019. A framework for collaborative learning in secure high-dimensional space. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 435–446.

[11] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. 2017. VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*. 1–8. https://doi.org/10.1109/ICRC.2017.8123650

[12] Mohsen Imani, Justin Morris, Samuel Bosch, Helen Shu, Giovanni De Micheli, and Tajana Rosing. 2019. AdaptHD: Adaptive Efficient Training for Brain-Inspired Hyperdimensional Computing. In *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 1–4. https://doi.org/10.1109/BIOCAS.2019.8918974

[13] Mohsen Imani, Saikishan Pampana, Saransh Gupta, Minxuan Zhou, Yeseong Kim, and Tajana Rosing. 2020. DUAL: Acceleration of Clustering Algorithms using Digital-based Processing In-Memory. In *53rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 356–371. https://doi.org/10.1109/MICRO50266.2020.00039

[14] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. 2018. Ecg heartbeat classification: A deep transferable representation. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*. IEEE, 443–444.

[15] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* 1, 2 (2009), 139–159.

[16] Jaeyoung Kang, Behnam Khaleghi, Yeseong Kim, and Tajana Rosing. 2022. XCelHD: An efficient GPU-powered hyperdimensional computing with parallelized training. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 220–225.

[17] Jiseung Kim, Hyunsei Lee, Mohsen Imani, and Yeseong Kim. 2023. Efficient Hyperdimensional Learning with Trainable, Quantizable, and Holistic Data Representation. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–6. https://doi.org/10.23919/DATE56975.2023.10137134

[18] Yeseong Kim, Mohsen Imani, Niema Moshiri, and Tajana Rosing. 2020. Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 115–120.

[19] Yeseong Kim, Mohsen Imani, and Tajana Rosing. 2017. ORCHARD: Visual object recognition accelerator based on approximate in-memory processing. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 25–32. https://doi.org/10.1109/ICCAD.2017.8203756

[20] Yeseong Kim, Mohsen Imani, and Tajana S Rosing. 2018. Efficient human activity recognition using hyperdimensional computing. In *Proceedings of the 8th International Conference on the Internet of Things*. 1–6.
[21] Yeseong Kim, Jiseung Kim, and Mohsen Imani. 2021. CascadeHD: Efficient Many-Class Learning Framework Using Hyperdimensional Computing. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 775–780.
[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[23] Denis Kleyko, Dmitri A Rachkovskij, Evgeny Osipov, and Abbas Rahim. 2021. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part II: Applications, Cognitive Models, and Challenges. *arXiv preprint arXiv:2112.15424* (2021).
[24] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).
[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (1998).
[26] Dehua Liang, Jun Shiomi, Noriyuki Miura, and Hiromitsu Awano. 2022. DistriHD: A Memory Efficient Distributed Binary Hyperdimensional Computing Architecture for Image Classification. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 43–49. https://doi.org/10.1109/ASP-DAC52403.2022.9712589
[27] Charles Mallah, James Cope, James Orwell, et al. 2013. Plant leaf classification using probabilistic integration of shape, texture and margin features. *Signal Processing, Pattern Recognition and Applications* 5, 1 (2013), 45–54.
[28] Nathan McDonald, Richard Davis, Lisa Loomis, and Johan Kopra. 2021. Aspects of hyperdimensional computing for robotics: transfer learning, cloning, extraneous sensors, and network topology. In *Disruptive Technologies in Information Sciences V*, Misty Blowers, Russell D. Hall, and Venkateswara R. Dasari (Eds.), Vol. 11751. International Society for Optics and Photonics, SPIE, 117510C. https://doi.org/10.1117/12.2585772
[29] Alisha Menon, Anirudh Natarajan, Laura I. Galindez Olascoaga, Youbin Kim, Braeden Benedict, and Jan M. Rabaey. 2022. On the Role of Hyperdimensional Computing for Behavioral Prioritization in Reactive Robot Navigation Tasks. In *2022 International Conference on Robotics and Automation (ICRA)*. 7335–7341. https://doi.org/10.1109/ICRA46639.2022.9811939
[30] Alisha Menon, Laura I. Galindez Olascoaga, Vamshi Balanaga, Anirudh Natarajan, Jennifer Ruffing, Ryan Ardalan, and Jan M. Rabaey. 2023. Shared Control of Assistive Robots through User-intent Prediction and Hyperdimensional Recall of Reactive Behavior. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 12638–12644. https://doi.org/10.1109/ICRA48891.2023.10161509
[31] Mehran Shoushtari Moghadam, Sercan Aygun, and M. Hassan Najafi. 2023. No-Multiplication Deterministic Hyperdimensional Encoding for Resource-Constrained Devices. *IEEE Embedded Systems Letters* 15, 4 (2023), 210–213. https://doi.org/10.1109/LES.2023.3298732
[32] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295* (2021).
[33] Yurii Evgen'evich Nesterov. 1983. A method of solving a convex programming problem with convergence rate O\bigl(k^2\bigr). In *Doklady Akademii Nauk*, Vol. 269. Russian Academy of Sciences, 543–547.
[34] P. Neubert and S. Schubert. 2021. Hyperdimensional computing as a framework for systematic aggregation of image descriptors. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 16933–16942. https://doi.org/10.1109/CVPR46437.2021.01666
[35] Peer Neubert, Stefan Schubert, and Peter Protzel. 2019. An introduction to hyperdimensional computing for robotics. *KI-Künstliche Intelligenz* 33, 4 (2019), 319–330.
[36] Yang Ni, Yeseong Kim, Tajana Rosing, and Mohsen Imani. 2022. Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 292–297.
[37] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks* 12, 1 (1999), 145–151.
[38] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers*. IEEE, 108–109.
[39] Jorge-L Reyes-Ortiz, Luca Oneto, Albert Sama, Xavier Parra, and Davide Anguita. 2016. Transition-aware human activity recognition using smartphones. *Neurocomputing* 171 (2016), 754–767.
[40] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.
[41] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).
[42] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report* 6 (2012).
[43] Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).
[44] Quanling Zhao, Kai Lee, Jeffrey Liu, Muhammad Huzaifa, Xiaofan Yu, and Tajana Rosing. 2022. FedHD: federated learning with hyperdimensional computing. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 791–793.
[45] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2018. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. arXiv:1606.06160 [cs.NE]
[46] Zhuowen Zou, Yeseong Kim, Farhad Imani, Haleh Alimohamadi, Rosario Cammarota, and Mohsen Imani. 2021. Scalable Edge-Based Hyperdimensional Learning System with Brain-like Neural Adaptation. In *Proceedings of the International Conference for High*

Performance Computing, Networking, Storage and Analysis (St. Louis, Missouri) *(SC '21)*. ACM, New York, NY, USA, Article 38, 15 pages. https://doi.org/10.1145/3458817.3480958

[47] Zhuowen Zou, Yeseong Kim, M Hassan Najafiψ, and Mohsen Imani. 2021. Manihd: Efficient hyper-dimensional learning using manifold trainable encoder. *DATE, IEEE* (2021).

Received 6 December 2023; revised 7 March 2024; accepted 13 May 2024