Efficient Forward-Only Training for Brain-Inspired Hyperdimensional Computing

Hyunsei Lee[1], Jiseung Kim[1], Seohyun Kim[1], Hyukjun Kwon[1], Mohsen Imani[2], Ilhong Suh[3] and Yeseong Kim[1]
[1]DGIST, [2]University of California Irvine, [3]COGA-ROBOTICS
{wwhslee, js980408, selenium, durwjsdnehd3523, yeseongkim}@dgist.ac.kr, m.imani@uci.edu, ihsuh@coga-robotics.com

***Abstract*—Hyperdimensional (HD) computing is an emerging paradigm inspired by human cognition, utilizing high-dimensional vectors to represent and learn information in a lightweight manner based on its simple and efficient operations. In HD-based learning frameworks, the encoding of the high dimensional representations is the most contributing procedure to accuracy and efficiency. However, throughout HD computing's history, the encoder has largely remained *static*, which leads to sub-optimal hypervector representations and excessive dimensionality requirements. In this paper, we propose novel forward-only training methods for HD encoders, Stochastic Error Projection (SEP) and Input Modulated Projection (IMP), which dynamically adjust the encoding process during training. Our methods achieve accuracies comparable to state-of-the-art HD-based techniques, with SEP and IMP outperforming existing methods by 5.49% on average at a reduced dimensionality of $D = 3,000$. This reduction in dimensionality results in a 3.32× faster inference.**

***Index Terms*—Hyperdimensional Computing, Data Representation, HDC encoding**

I. Introduction

Hyperdimensional (HD) computing, also known as Vector Symbolic Architectures (VSAs), is a computing paradigm modeled after human long-term memory [1]. HD computing has recently been gaining recognition as an attractive alternative to traditional deep learning techniques primarily because of its lightweight computations. HD computing is a symbolic representation system that maps data points to high-dimension random vectors of distributed representation, called *hypervectors*; that is, hypervectors are extremely wide words with bit-widths in the thousands that represent data holistically over the entire word [2]. As efforts to push the processing of information closer to the application source continue to increase, HD computing is becoming a more promising solution for lightweight learning. The holistic nature of the HD representation system means elements in a hypervector are dimension-independent, allowing for implementations of efficient and highly parallel hardware accelerators, including GPGPU [3], FPGA [4], and ASIC designs [5].

HD learning models for classification tasks first map input data to hypervectors. Various encoding methods have been proposed [6], [7], but the overarching idea has been to generate vectors or matrices of random elements such that the outer product between it and input data would project the data to high-dimension holistic vectors. In many HD-based learning algorithms [6], [4], [8], hypervectors of the same class are then aggregated into a single hypervector, called *class hypervectors*, that represents the class through a universal pattern. The simplest form of an HD model takes inference data and compares its hypervector representation to the class hypervectors. The class hypervector that yields the highest similarity becomes the predicted label. The similarity of hypervectors is measured through metrics such as dot product and cosine similarity. Finally, further refinement of the HD model is achieved through a retraining procedure [9] that fine-tunes class hypervectors to increase their similarity to correct labels while decreasing similarity to incorrect ones.

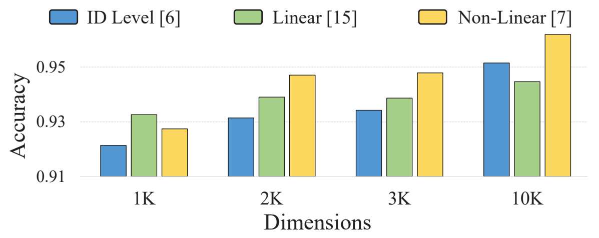


Fig. 1: Accuracies of Popular Encoders as Dimensions Increase. HDC Requires High Dimensions to Perform.

In our examination of current retraining procedures within HD learning models, we observe a significant limitation: the sole focus on *fine-tuning class hypervectors.* This approach operates under the assumption that the encoding process will yield sufficiently distinct hypervectors for data associated with different labels. However, this overlooks a critical aspect of HD computing, the role of *hypervector encoding* in determining the overall accuracy of the trained model. The projection matrices generated during the encoding phase remain *static* throughout both the initial training and subsequent retraining procedures, and, as a result, the resulting hypervector representations for any given data set are *immutable*. To address the limitations associated with static encoders in HD learning models, prior work has focused on either preprocessing the data [10] or integrating neural networks as inputs to the HD framework [11], [12]. While these approaches can improve performance, they introduce considerable computational overhead, and more critically, they fail to resolve the inherent static nature of the HD encoder.

This rigidity can lead to two critical issues: (i) sub-optimal hypervector representations, especially in scenarios involving complex data, which culminates in ineffective learning, and (ii) the use of excessive dimensions, e.g., $D = 10,000$, to compensate for the error incurred through ineffective projec-

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No.2022-0-00991, 1T-1C DRAM Array Based High-Bandwidth, Ultra-High Efficiency Processing-in-Memory Accelerator). This work was also supported by Cogarobotics.

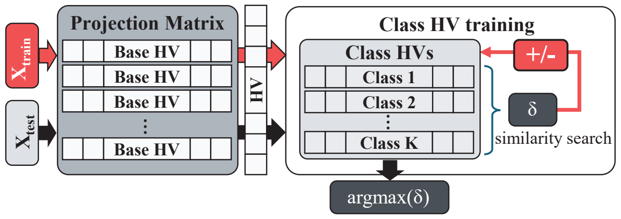


Fig. 2: Typical HD-Based Learning Procedure

tions of the raw data into high-dimensional space, resulting in a high computational burden to perform HD computing. We demonstrate in Figure 1 on a face recognition dataset (FACEA, see Table I) that the three most common encoding methods all require high dimensions for the best performance.

In this paper, we introduce two novel frameworks for HD encoder training designed to generate accurate hypervector representations learned during the training process. Adhering to the core philosophy of *brain-inspired* HD computing, we draw inspiration from forward-only training methods proposed in deep learning, a body of work that considers the biological plausibility of weight updates. Our proposed methods train hypervector representations with a significant reduction in dimensions while maintaining high accuracy.

We summarize our contributions as follows:

- We present *Stochastic Error Projection* (SEP), which provides implicit feedback to the encoder at each retraining iteration. Though it exhibits a slightly slower convergence rate, it maintains high efficiency and outperforms existing HD-based algorithms.
- We also introduce *Input Modulated Projection* (IMP). This algorithm first computes per-class error through the standard HD learning procedure and subsequently performs a second pass, modulating the input with the error to update the encoder. We observe that, despite requiring additional computational resources due to the encoding of two separate hypervectors, this method converges rapidly and shows high accuracy.
- We additionally include in this work analysis that explores the implications of our proposed algorithms on acceleration designs. Our experimental results show that training the HD encoder dynamically can significantly increase accuracy. The proposed algorithms outperform the state-of-the-art HD encoder training method based on backpropagation even at a lower dimension, e.g., 5.49% for $D = 3,000$, resulting in a $3.32\times$ faster inference on average.

II. Background and Related Work

A. Hyperdimensional Computing

1) Hyperdimensional Representation: Hyperdimensional (HD) computing is a computing paradigm inspired by sparse distributed memory (SDM) [1], a model of human long-term memory. The concept and mathematical foundations of the HD representation system are introduced in [2]. The key concept of HD representation is to represent data symbolically as *hypervectors*. Hypervectors are 1) high in dimensions, 2) holistic in representation, and 3) randomly generated. These properties of hypervectors allow orthogonality between unrelated data points and similarity between related ones. The relation of hypervectors is measured through a similarity metric; dot product similarity is a popular metric used in HD models and is the metric we will be using throughout this paper.

2) Manipulating Hypervectors: There are two operations for manipulating hypervectors. *Binding* $\otimes$ is implemented as an element-wise multiplication. The binding operation associates hypervectors and the resulting vector is orthogonal to its inputs. *Bundling* $\oplus$ is an element-wise addition. This operation combines hypervectors into a single hypervector representing the set of inputs. This hypervector is similar to its inputs.

3) HD-based Learning: The learning procedure using HD-based representation systems (shown in Figure 2) first encodes data into hypervectors, $\vec{H}$. For training, all hypervectors with the same label are bound together to form a single hypervector representing the label. These hypervectors are called the *class hypervectors*. The set of k class hypervectors is denoted as $\mathbb{C} = \{\vec{C}_1, \vec{C}_2, \cdots, \vec{C}_k\}$ where a class hypervector for the i^{th} label is computed as $\vec{C}_i = \vec{H}_1^i \oplus \vec{H}_2^i \oplus \cdots \oplus \vec{H}_j^i$, where j is the number of data samples for a given label. This simple accumulation of hypervectors of the same label is called single-pass training and is very fast and efficient but shows limited accuracy.

Therefore, a retraining procedure is further employed. Retraining is an iterative procedure where encoded hypervectors, $\vec{H}$, are measured for similarity against the set of class hypervectors, $\mathbb{C}$. If the similarity is closest to its true label, we leave the model as is. Otherwise, we adjust both the true label and misclassified class hypervectors by the encoded hypervector or its complement to make the correct class more similar while making the misclassified class more dissimilar.

$$\vec{C}_{\text{miss}} = \vec{C}_{\text{miss}} \oplus -\vec{H} \ ; \ \vec{C}_{\text{true}} = \vec{C}_{\text{true}} \oplus \vec{H} \qquad (1)$$

Finally, inference is done by encoding inference data through the same encoding procedure used during training. We call these query hypervectors, $\vec{Q}$, and they are compared for similarity with each class hypervectors. The class with the highest similarity is our inferred label, i.e., $argmax(\delta(\mathbb{C}, \vec{H}))$, where k is the number of classes and δ is our similarity metric.

B. Encoders in HDC

Encoding is the method in which data are converted to hypervectors. The encoding phase of the HD classification workflow is the most important as it is the most contributing factor to accuracy.

1) Static Encoders: Virtually all encoding algorithms (except [13] discussed later) proposed in the literature use static encoders. *ID-level encoding* [6], [14] generates binary/bipolar hypervectors for data feature values and positions. *Random projection encoding* [15], [16] generates a projection hypermatrix of binary/bipolar elements. This encoding method directly binds feature values to the projection matrix, removing the need for separate value hypervectors. *Nonlinear encoding* [7], [17], [9] is cited as the best performing encoding algorithm. It is algorithmically identical to random projection. It differs only in that it draws its projection matrix elements from Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$.

Note that these methods do not utilize knowledge obtainable from training samples. Once generated, there is nothing in the learning framework that enhances the quality of representations. As a result, these algorithms require hypervectors of much higher dimensions.

2) TrainableHD [13]: Recently, TrainableHD was proposed as an HD learning framework with a dynamic encoder. It is based on the nonlinear encoding method using MASS retraining [9] that updates each class hypervector based on class-wise similarity differences. TrainableHD updates the projection matrix using per-feature error hypervectors and is theoretically based on backpropagation used in deep learning. The base algorithm, however, is computationally expensive, and backpropagation requires the transport of synaptic weight information, making it biologically implausible [18].

C. Biologically Inspired Alternatives to Backpropagation

In the domain of neural networks, several training paradigms have been proposed to address the biologically implausible elements associated with traditional backpropagation. In this context, we offer a brief summary of research initiatives, with a particular focus on frameworks that can be classified as forward-only algorithms. *Feedback alignment* (FA) [19] propagates random fixed connections. The random matrices deliver useful modulatory information as the forward connections are driven to align with the feedback. *Direct feedback alignment* (DFA) [20] extends the FA algorithm by propagating random connectivity matrices to each hidden layer. Finally, *Present the Error to Perturb the Input to modulate Activity* (PEPITA) [21] perturbs input data based on error-related information. The backward pass is replaced with a second forward pass with the perturbed input. In this work, we take these ideas, carefully modify and integrate them into the HD learning process such that HD computing can learn from *dynamic* representations.

III. Forward-only HD Encoder Training

A. Overview

In this paper, we explore the training of the projection matrices, or base hypervectors, through methods inspired by forward-only algorithms. This approach contrasts with much of the existing literature [6], [15], [7], where projection matrices are static, as well as a recent work [13] employing a backpropagation-based HD encoder training technique. Figure 3 shows an overview of the HD learning process, including the proposed forward-only HD encoder training procedures. Our algorithm initiates the training workflow with the conventional class hypervector retraining procedure, specifically adhering to the nonlinear encoding method [7] and the MASS retraining algorithm [9], which fine-tunes class hypervectors based on their calculated errors based on hypervector similarity measurements.

After updating the class hypervector with a sample or a mini batch of samples, we proceed to update the projection matrices. To this end, we introduce two forward-only training methods, SEP and IMP. In SEP, we focus solely on the observed error in the class hypervectors without considering the properties of the raw input data or the encoded hypervectors. This error is quantified using the similarity values between the class hypervectors and the encoded hypervectors; if the current model produces highly undesirable results in terms of

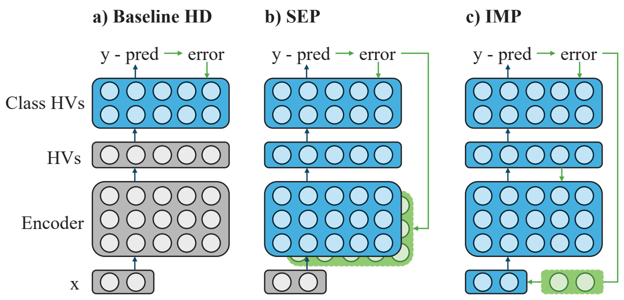


Fig. 3: High-level Comparison of Proposed Methods with Conventional HDC

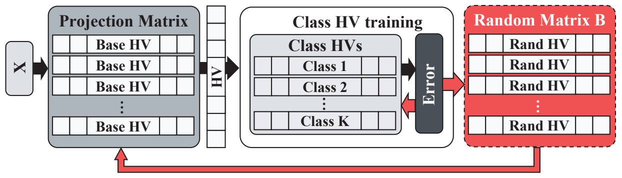


Fig. 4: Overview of the SEP Procedure

the class-wise similarity, it is reflected in a higher error value that elicits more updates to the projection matrices. In contrast, IMP explicitly incorporates the encoded hypervector with the projection matrices. We modulate the encoded samples with the degree of measured error to compute the magnitude of updates applied to the projection matrices. The two methodologies each come with their own advantages and drawbacks, resulting in distinct learning trajectories which we cover in the following sections.

B. Stochastic Error Projection

1) Stochastic Error Projection (SEP): We aim to develop *dynamic* encoders for HD computing that bring out the full potential of HD training while maintaining HD computing's biological fidelity. In this section, we propose *Stochastic Error Projection* (SEP). The fundamental intuition behind SEP is premised on the idea that *substantial updates to the base hypervectors* are needed when the current model displays a *high classification error*.

Figure 4 shows an overview of the training procedure. Unlike existing methods such as the MASS training algorithm [9], which focuses on the classification errors that originate from inaccuracies in the class hypervectors, we further consider the contribution to inaccuracies caused by erroneous base hypervectors. In SEP, the magnitude of adjustments to each element of the base hypervectors is stochastically determined; that is, elements are perturbed with a higher degree of randomness when elevated errors are observed for current samples. In contrast, minimal adjustments are made to the base hypervectors when the overall classification error is low. This is done under the assumption that they are adequately accurate. While the SEP approach bears similarities to the DFA [20] method proposed for deep learning, we introduce a modification to address a critical shortcoming of DFA, i.e., its tendency to overfit, thereby optimizing its efficiency with higher robustness within the domain of HD computing. Algorithm 1 details the SEP procedure.

Algorithm 1 Stochastic Error Projection (SEP)

```
1: for data $\mathbf{x}$ in a training dataset do
2:     // Ⓐ Encoding
3:     $\vec{H} \leftarrow sign(cos(\mathbb{P} \otimes \mathbf{x}))$
4:     // Ⓑ Update class hypervectors $\mathbb{C}$
5:     $\vec{s} \leftarrow softmax(\delta(\mathbb{C}, \vec{H}))$
6:     $\vec{e} \leftarrow y - \vec{s}$
7:     $\mathbb{C} \leftarrow \mathbb{C} \oplus \lambda \vec{H} \vec{e}$
8:     // Ⓒ Update projection matrix $\mathbb{P}$
9:     updates $\leftarrow$ mean($\vec{e}$) $\cdot \lambda \cdot \mathbb{B}$
10:    $\mathbb{P} \leftarrow \mathbb{P} \oplus$ updates
11: end for
```

Given a data sample $\mathbf{x} = \{v_1, v_2, \cdots, v_F\}$, where F is the number of feature values, the projection matrix is generated as $\mathbb{P} = \{\vec{P}_1, \vec{P}_2, \cdots, \vec{P}_F\}^D$, where D is dimensions and each $\vec{P}_i$ is drawn from Gaussian distribution. Encoding (Ⓐ) is computed as $\vec{H} = \text{sign}(\cos(v_1 \times \vec{P}_1 + v_2 \times \vec{P}_2 + \cdots + v_F \times \vec{P}_F)$. For the retraining procedure (Ⓑ), encoded hypervectors, $\vec{H}$, are compared for similarity with class hypervectors, $\mathbb{C}$, and normalized with the *softmax* function. We can then obtain the per-class error, $\vec{e}$, with the ground truth one hot encoding, y, and normalized similarities, $\vec{s}$. The class hypervectors are updated by binding the encoded hypervector scaled with the per-class errors and a learning rate, λ. This so far aligns to the widely used nonlinear encoding [7].

To update our projection matrix (Ⓒ), we generate an additional hypermatrix, $\mathbb{B}$, with the same dimensions as $\mathbb{P}$. The $\mathbb{B}$ hypermatrix is initialized with He uniform distribution [22], $\mathbb{B} \sim \mathcal{U}(-\sqrt{\frac{6}{F}}, +\sqrt{\frac{6}{F}})$, where F is the number of features. This random $\mathbb{B}$ matrix can be fixed for subsequent iterations of retraining. To update the encoder, we simply bind to $\mathbb{P}$ the $\mathbb{B}$ hypermatrix scaled with the per-class errors averaged and λ, the learning rate.

2) Error Projection: Fixed vs. Variable: In the backpropagation-based TrainableHD [13], the projection matrix is given explicit feedback through class errors. However, in SEP, the random hypermatrix $\mathbb{B}$ presents implicit feedback to the encoder with only an approximate update direction and angle through averaged errors, $\vec{e}$. A *fixed* $\mathbb{B}$ matrix pushes updates to be guided by changes in the projection matrix $\mathbb{P}$. Although convergence, especially during earlier iterations of training, may be slower than giving explicit feedback, the updates quickly improve to be on par with explicit updates.

Another method of implementing the $\mathbb{B}$ hypermatrix is newly generating the random matrix at every iteration, unlike DFA [20]. Because the matrix is not fixed, it is slightly less sensitive to changes in the projection matrix and, therefore, is slower to converge. However, with enough iterations, this method is more robust and still manages to reach the same performance as the fixed variant of SEP at the end of training.

C. Input Modulated Projection

In this section, we present *Input Modulated Projection* (IMP). The guiding intuition behind IMP is the understanding that there are two sources of errors during the training process: first, the inaccuracies in the representation of the projection matrices, a focus also shared by the previously discussed SEP algorithm; and second, the resulting inaccuracies in the high-dimension representations of the input samples themselves.

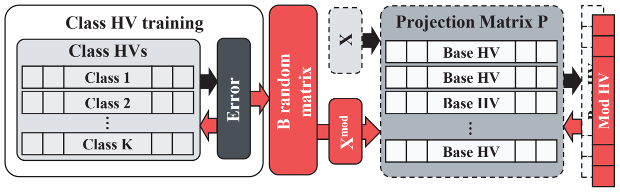


Fig. 5: Overview of the IMP Procedure

IMP incorporates in its learning procedure the training of a dynamic encoder by utilizing the two sources of errors. It is worth noting that the IMP algorithm can be construed as a specialized variant of the PEPITA algorithm with the extension of its applicability into the context of HD computing.

Algorithm 2 shows IMP's training procedure. IMP first encodes input data into hypervectors ($\vec{H}$), then calculates and normalizes similarities ($\vec{s}$), and finally produces per-class errors ($\vec{e}$) that are used to update the class hypervectors ($\mathbb{C}$). The procedure up until this point is identical to that of SEP.

Algorithm 2 Input Modulated Projection (IMP)

```
1: for data $\mathbf{x}$ in a training dataset do
2:     // Ⓐ first pass
3:     $\vec{H} \leftarrow sign(cos(\mathbb{P} \otimes \mathbf{x}))$
4:     $\vec{s} \leftarrow softmax(\delta(\mathbb{C}, \vec{H}))$
5:     $\vec{e} \leftarrow y - \vec{s}$
6:     $\mathbb{C} \leftarrow \mathbb{C} \oplus \lambda \vec{H} \vec{e}$
7:     // second pass
8:     // Ⓑ Modulate $\mathbf{x}$
9:     $\mathbf{x}_{mod} \leftarrow \mathbf{x} + (\vec{e} \times \mathbb{B})$
10:    // Ⓒ Encode modulated $\vec{H}$
11:    $\vec{H}_{mod} \leftarrow \mathbb{P} \otimes x_{mod}$
12:    // Ⓓ Update projection matrix $\mathbb{P}$
13:    updates $\leftarrow \mathbf{x} \otimes (\vec{H} - \vec{H}_{mod})$
14:    $\mathbb{P} \leftarrow \mathbb{P} \oplus$ (updates $\times \lambda$)
15: end for
```

Instead of sending feedback to the projection matrix $\mathbb{P}$, IMP performs a second pass to update the encoder. Figure 5 illustrates IMP's procedure at its second pass. The first pass (Ⓐ) exists to compute the error and update the class hypervectors. The goal in (Ⓑ) is to modulate the input $\mathbf{x}$ with the per-class errors, $\vec{e}$, computed in the first pass. To this end, we generate a fixed random matrix $\mathbb{B}$ of $k \times F$ dimensions, where k is the number of classes (labels) and F is data features. This matrix samples elements from He uniform distribution [22] and is designed to project $\vec{e}$ to match the input $\mathbf{x}$'s dimensions, through which we can construct the modulated input $\mathbf{x}_{mod} \leftarrow \mathbf{x} + (\vec{e} \times \mathbb{B})$. To update our projection matrix, we first encode (Ⓒ) the *modulated input* through the same encoding process to obtain the error-modulated hypervector, $\vec{H}_{mod}$. We can now update the projection matrix (Ⓓ) by binding the original input with a hypervector that is given by the difference in activation of the hypervector encoded from raw data, $\vec{H}$, and the error modulated hypervector, $\vec{H}_{mod}$.

IV. Accelerator Implementation

A. Acceleration Strategy of Forward-Only Encoder Training

In HD computing, data are represented through vectors of high dimensions, requiring operations such as matrix multiplication and element-wise operations. The data-independent nature between elements allows for operations that are inherently suited for parallel processing architectures, such as FPGAs and GPUs, which can significantly enhance throughput and reduce latency. For instance, as illustrated in Figure 6a, our FPGA implementation utilizes a processing engine (PE)-based design where the host communication occurs via the AXI4 stream to initiate operations while resulting hypervectors are stored in DRAM. Each PE performs highly parallel HD operations using state-of-the-art implementation strategies, such as matrix multiplication for forward-path encoding and similarity search, and vector additions for class hypervector updates, input modulations in IMP, etc.

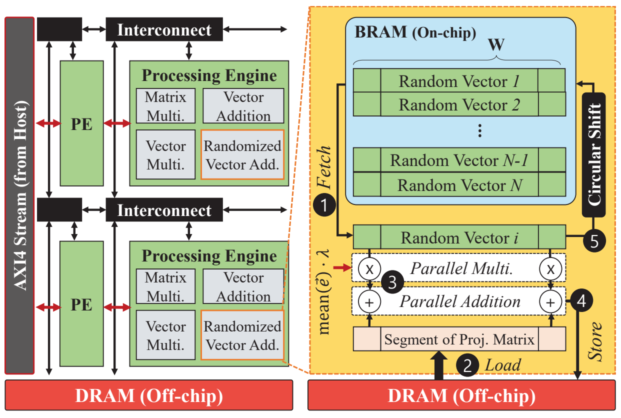
Here, the challenge is not in the complexity of the operations themselves but in optimizing data transfer and computational efficiency to leverage the full potential of these architectures. One of the primary challenges in accelerating HD operations is the memory-bound nature of these tasks. As the dimension of the data increases, the volume of data that must be transferred between memory and processors can become a significant bottleneck. To address this, we optimize memory access patterns and enhance parallel execution capabilities for specific HD operations introduced in our work. SEP, for example, incorporates stochastic elements into its training process, requiring sophisticated memory management to handle the random elements efficiently during computation. Our design reduces the frequency and volume of global memory accesses required during computation by effectively using shared memory on GPUs. Similarly, FPGA leverages configurable logic blocks and dedicated on-chip memory resources to streamline the execution of repetitive and parallelizable tasks. In the following sections, we discuss the details of our accelerator designs for the proposed encoder training methods.

B. Optimization Strategies for SEP

The SEP algorithm introduces a unique computational challenge in Hyperdimensional computing: the integration of stochastic elements into the projection matrix during training. This process requires not just basic operations but the addition of a dynamic component, which is the random variable scaled by the mean error and learning rate, expressed as $mean(\vec{e}) \cdot \lambda \cdot r$. Such operations are inherently data-intensive and require efficient memory management and computational strategies. In naive implementations of the SEP algorithm, a large random matrix, matching the size of the projection matrix, would be pre-generated and stored. During each training iteration, both the projection matrix and this random matrix need to be read, which doubles the memory bandwidth requirements and can quickly become a bottleneck in systems where memory access speeds are a limiting factor. We address this issue both on the FPGA and GPU using specialized operations for the variable SEP algorithm discussed in Section III-B2.

FPGA Implementation For the SEP algorithm's implementation on FPGA platforms, an advanced approach is utilized to manage the stochastic component of the algorithm efficiently. This approach centers around the use of Block RAM (BRAM) for dynamically managing random vectors that are crucial for the stochastic updates to the projection matrix.

Figure 6b describes this implementation strategy, which is implemented in the randomized vector addition logic. Here, BRAM is configured to store a set of random vectors, configured as an $N \times W$ array, where N represents the number



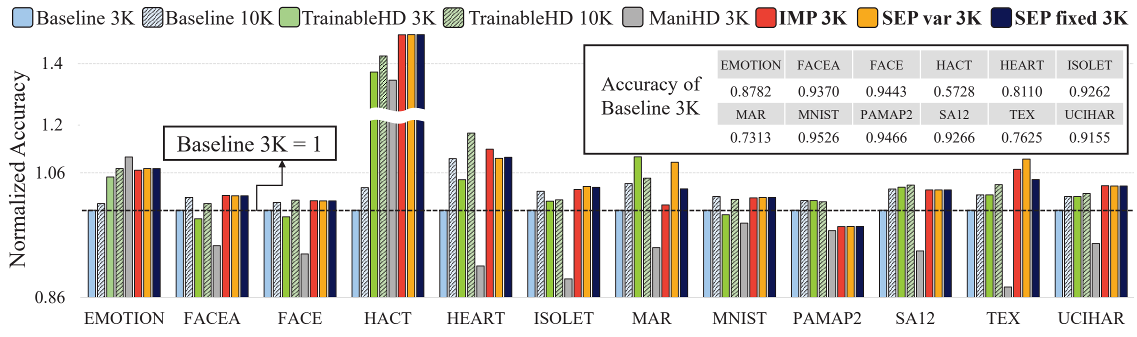
(a) Implementation Overview (b) Projection Matrix Update in SEP

Fig. 6: FPGA Acceleration Implementation

of rows and W the width of each row. Each row in the BRAM is aligned with the width of data chunks read from the DRAM at each cycle. This alignment ensures that data fetched from DRAM and the corresponding random vectors from BRAM can be processed in parallel. The process begins by fetching a random vector from the BRAM (❶). Concurrently, a segment of the projection matrix is retrieved from the DRAM (❷). These two vectors are then added together in a parallel processing step by multiplying the scaling scalar factor (i.e., $mean(\vec{e}) \cdot \lambda$) to the fetched random values (❸). The computed results, i.e., the updated segment of the projection matrix, are then stored back to the DRAM (❹).

At the same time, the random vector fetched from BRAM undergoes a circular (rotation) shift, and it is written back into the same BRAM row (❺). This shift alters the vector, ensuring that when the next time it is accessed, it appears as a newly sampled random vector since each random element is independently sampled. It allows us to reuse the stored random vector up to $N \times W$ times to obtain different random vectors, guaranteeing that the random vectors remain varied while adhering to the stochastic nature required by the SEP algorithm. Once we reach the end of the reusing cycles, we reload a new set of random vectors from the off-chip DRAM, but this occurs infrequently, making it a minor part of the overall execution timeline as it can be amortized over many cycles of computations. This strategic use of BRAM reduces the memory bandwidth requirements for updating the projection matrix by approximately half – since we do not need to load the random hypervector from DRAM with the projection matrix of the same size every time.

GPU Implementation For GPU implementations, we utilize the random number generation library, cuRAND, to generate random numbers directly within the GPU's processing units on the fly. In SEP's GPU adaptation, the focus is on integrating the generation of random numbers into the computational kernels. This integration allows for a reduction in memory transfers between the GPU streaming cores and the global DRAM memory in a similar fashion to the FPGA. To this end, we implement a customized CUDA kernel, which is integrated as an operation in the PyTorch library.



Accuracy of Baseline 3K

EMOTION	FACEA	FACE	HACT	HEART	ISOLET
0.8782	0.9370	0.9443	0.5728	0.8110	0.9262
MAR	**MNIST**	**PAMAP2**	**SA12**	**TEX**	**UCIHAR**
0.7313	0.9526	0.9466	0.9266	0.7625	0.9155

Fig. 7: Accuracy Comparison

TABLE I: Evalution Datasets

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

C. Optimization Strategies for IMP

While the IMP method incorporates an additional layer of complexity compared to SEP, the FPGA and GPU platforms also parallelize the extra operational demands efficiently. For example, the encoding procedure of the modulated input $\mathbf{x}$ can also be carried by the matrix multiplication in the parallelized FPGA, in our case, using a systolic array structure.

However, since IMP requires the manipulation of both the original encoded hypervector, $\vec{H}$, and its modulated variant, $\vec{H}_{mod}$, the computational demands increase, particularly during the phase where $\mathbf{x} + \vec{e} \times \mathbb{B}$ is encoded to produce $\vec{H}_{mod}$. An effective strategy employed in both FPGA and GPU implementations involves temporarily storing $\vec{H}$ after its initial computation and reusing it for subsequent operations. This approach minimizes the redundant recomputation of $\vec{H}$, thereby conserving computational resources and reducing execution times. On FPGAs, this can be achieved by utilizing the on-chip memory to store $\vec{H}$, while GPUs can keep $\vec{H}$ in the high-bandwidth GDDR memory.

Although IMP inherently places greater demands on computational and memory resources, the advanced capabilities of FPGA and GPU architectures help in effectively managing these challenges. It is also worth noting that despite its higher initial resource usage, IMP's sophisticated approach to error estimation—taking into account the current input samples—can lead to faster convergence in the training phase, potentially offsetting the additional costs with more rapid improvements in learning accuracy.

V. Experimental Results

A. Experimental Setup

For the experimental setup of our study running on GPUs, we implement all HD learning procedures on NVIDIA GeForce RTX 2080 SUPER GPU with an Intel Core i9-10900K CPU using the PyTorch 1.12 framework. We also implemented a custom CUDA kernel designed specifically for the SEP operations, which enables memory-efficient parallel processing, and integrated it as an extension to PyTorch. To implement the FPGA-based accelerator, we utilized Vitis HLS 2023.2 and extended the Vitis BLAS L2/L3 library to support other parallel element-wise operations and optimized HD operations discussed in Section IV. These computations were then deployed and tested on a Xilinx U200 FPGA card under a four PE setting, which operated at a 200 MHz clock frequency.

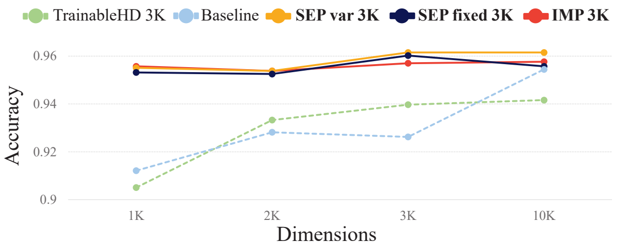


Fig. 8: Accuracies at Various Dimensions in ISOLET

We evaluate IMP and two versions of the SEP algorithm, one with a fixed $\mathbb{B}$ (SEP fixed) and another with a newly generated $\mathbb{B}$ at each iteration (SEP var). For comparison, we chose static encoding techniques: nonlinear encoding [7] as it is widely used for its high accuracy and ManiHD [23] a framework that implements manifold projections before the static HD encoder. We chose ManiHD for accuracy comparisons as we feel it represents the efforts in HD research to increase accuracy while limited by static encoders. In addition, we compare our work with TrainableHD [13], which uses a backpropagation-like method to implement a dynamic encoder. Each model is evaluated based on 50 retraining iterations. The datasets used to assess the models are listed in Table I.

B. Classification Accuracy

Figure 7 compares the accuracies of various training algorithms. In this evaluation, the results for the Baseline [7] and TrainableHD [13] are shown at both 3K and 10K dimensions, while our proposed methods—SEP var, SEP fixed, and IMP—are evaluated at 3K dimensions. Remarkably, all three methods achieve performances comparable to Train-

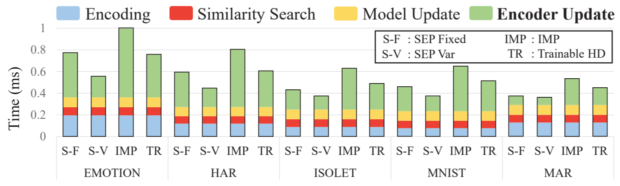


Fig. 9: Training Speed over Different Datasets on GPU

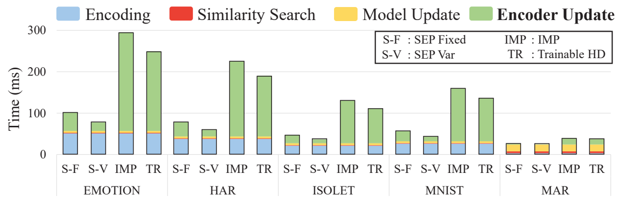


Fig. 10: Training Speed over Different Datasets on FPGA

ableHD at 10K dimensions and consistently outperform Baseline, ManiHD [23], and TrainableHD at 3K dimensions. The accuracy is 5.49% higher on average, and the enhanced performance of our methods can primarily be attributed to their handling of errors during the training process. The SEP and IMP algorithms can recognize and adjust not only class-specific errors but also broader errors resulting from suboptimal hypervector encoding. This dual awareness allows for a more refined tuning of the encoder, leading to the generation of more accurate and compact hypervector representations.

Furthermore, as demonstrated in Figure 8 for the ISOLET dataset, we observe that the hypervector representations produced through the proposed methods not only allow for compact storage but also maintain a consistent accuracy across varying dimensions. This property directly influences the efficiency and accuracy of the learning model, making the proposed methods particularly advantageous for practical applications where computational resources and memory are limited.

C. Computational Efficiency

Figure 9 and Figure 10 show training times for a single mini-batch (32) of data on the NVIDIA GeForce RTX 2080 SUPER GPU and the Xilinx U200 FPGA, respectively. The computation timings shown are the averaged times of all batches throughout all training epochs. In the two Figures, *Encoding, Similarity Search,* and *Model Update* are virtually identical for the four models as they all follow the same procedure. The timings reported for Figure 9 and 10 will, therefore, only be the time taken to compute encoder updates.

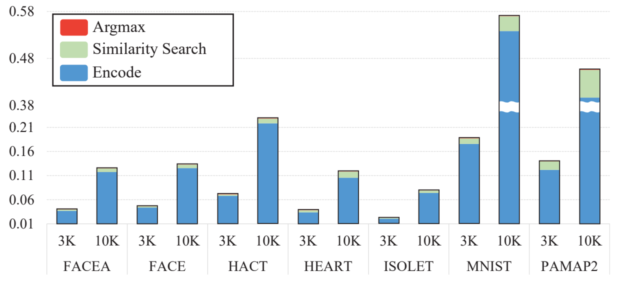


Fig. 11: GPU Inference Time (s) for 3K and 10K Dimensions

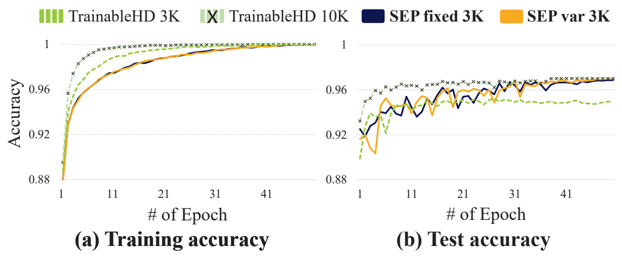


Fig. 12: SEP Accuracy Changes over Iterations

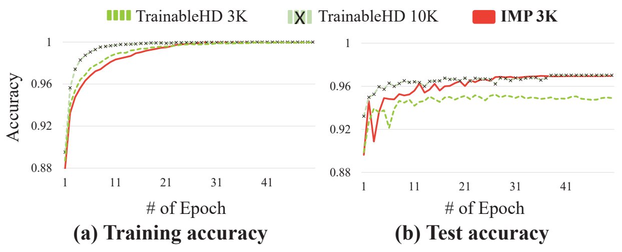


Fig. 13: IMP Accuracy Changes over Iterations

Both SEP fixed and SEP var have fewer computations than TrainableHD [13] and show a relatively lightweight training process. Across all datasets, in our GPU implementation, SEP var was up to 2.04× faster than TrainableHD and was 2.18× faster on average. In our FPGA implementation SEP var was 9.54× faster on average and, on the MAR dataset, was 13.33× faster. SEP fixed performed up to 2.03× and 5.56× faster on the GPU and FPGA, respectively. The higher efficiency stems from the characteristics of the SEP algorithm, which does not need the complex re-encoding procedure for a given input feature vector like the IMP procedure. Also, SEP var pushes optimization beyond the fixed variation as it reduces the significant overhead from memory communication by generating the random values spontaneously on GPUs or reusing the stored data in on-chip FPGA BRAMs.

IMP, however, uses a second pass with the modulated input during training and needs to encode a second hypervector. This, unfortunately, adds extra computation to the procedure. Nevertheless, even with significantly more computations, IMP was 52.15% slower than TrainableHD on the GPU and, at its best, was only 12.27% slower on the FPGA.

In practice, however, our proposed models are much more efficient than existing techniques for *inference* because they achieve similar accuracy to that of TrainableHD 10K at much lower dimensions. Figure 11 shows the timings of the inference procedure for 3K and 10K dimensions. Like TrainableHD, our proposed methods use the identical inference procedure as a standard HD model. In our measurements, an HD model using $D = 3,000$ on GPUs is on average 3.32× faster than a model with $D = 10,000$.

D. Learning Performance and Convergence

SEP Fixed / Variable Here, we present how SEP learns through implicit feedback. Figure 12 shows accuracy changes during training and inference for each retraining epoch. As expected, both fixed and variable versions of SEP start off with seemingly random updates, and in the very initial iterations, this is even more so with SEP var. This behavior can be attributed to the stochastic nature of the updates in SEP, where the projection matrix is perturbed using a random matrix scaled by the mean error. In the early stages of training, the model has not yet learned meaningful representations, leading to high errors and, consequently, larger, more random perturbations to the projection matrix.

However, as iterations pass, both SEP models quickly find representations that best-fit data features and show performance comparable to that of TrainableHD 10K. This rapid improvement in accuracy indicates that the implicit feedback provided by the error-projected matrix effectively guides the model toward learning more suitable representations. The stochastic perturbations allow the model to explore the representation space and converge to a solution that minimizes the classification error.

SEP var and SEP fixed show final inference accuracies of 96.99% and 96.87% while TrainableHD 3K and 10K results in 94.91% and 97.02% accuracy, respectively. These results demonstrate that SEP is able to achieve competitive performance with TrainableHD 10K by learning effective representations through implicit feedback, despite using a simpler and more computationally efficient approach. The key advantage of SEP lies in its ability to train the encoder using only forward passes and stochastic updates, eliminating the need for explicit backpropagation-like updates while still achieving high accuracy.

IMP Figure 13 shows the results during learning with the IMP algorithm. This model has a comparable number of computations to TrainableHD and, therefore, is not as efficient as SEP. However, it converges much faster than both SEP algorithms and performs just as well (96.97%), meaning IMP requires fewer training iterations than SEP.

The faster convergence of IMP can be explained by its explicit incorporation of the encoded hypervector in the update process. By modulating the input with the error and performing a second pass to update the projection matrix, IMP directly adjusts the encoder based on the discrepancy between the original and error-modulated hypervectors. This targeted update mechanism allows IMP to rapidly improve the representation quality and minimize classification errors.

The results show that both SEP and IMP are able to continually find better representations throughout their training procedures, while TrainableHD quickly converges but remains at worse representations. TrainableHD's explicit feedback based on training errors may lead to overfitting, as the model becomes too specialized to the training data and fails to generalize well. In contrast, SEP and IMP's forward-only training approach, with implicit and modulated feedback, allows them to learn more robust and generalizable representations.

VI. Conclusion

In this paper, we proposed two frameworks, Stochastic Error Projection (SEP) and Input Modulated Projection (IMP), to extend HD computing's learning framework to encompass the training of the encoder. SEP leverages observed training errors to provide implicit feedback, optimizing encoder performance without direct error gradients. IMP utilizes a second pass that utilizes error-projected hypervectors for dynamic adjustments to the encoding matrix. Our evaluations show that both methods successfully yield better representations and, thus, are able to achieve state-of-the-art accuracies at significantly lower dimensions. In addition, hardware implementations of SEP Variable and Fixed on FPGAs showed an average of 9.54× and 4.38× faster training time to that of existing techniques at equal dimensions.

References

[1] Pentti Kanerva. *Sparse distributed memory*. MIT press, 1988.
[2] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1(2):139–159, 2009.
[3] Jaeyoung Kang, et al. Xcelhd: An efficient gpu-powered hyperdimensional computing with parallelized training. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 220–225. IEEE, 2022.
[4] Mohsen Imani, et al. Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 493–498, 2019.
[5] Sohum Datta, et al. A programmable hyper-dimensional processor architecture for human-centric iot. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(3):439–452, 2019.
[6] Mohsen Imani, et al. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, 2017.
[7] Mohsen Imani, et al. Dual: Acceleration of clustering algorithms using digital-based processing in-memory. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 356–371, 2020.
[8] Arpan Dutta, et al. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. New York, NY, USA, 2022. Association for Computing Machinery.
[9] Yeseong Kim, et al. Cascadehd: Efficient many-class learning framework using hyperdimensional computing. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 775–780, 2021.
[10] Mohsen Imani, et al. Neural computation for robust and holographic face detection. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, page 31–36, New York, NY, USA, 2022. Association for Computing Machinery.
[11] Peer Neubert, et al. An introduction to hyperdimensional computing for robotics. *KI - Künstliche Intelligenz*, 33(4):319–330, Dec 2019.
[12] Peter Sutor, et al. Gluing neural networks symbolically through hyperdimensional computing, 05 2022.
[13] Jiseung Kim, et al. Efficient hyperdimensional learning with trainable, quantizable, and holistic data representation. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2023.
[14] Cheng-Yen Hsieh, et al. Fl-hdc: Hyperdimensional computing design for the application of federated learning. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–5, 2021.
[15] Mohsen Imani, et al. A framework for collaborative learning in secure high-dimensional space. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 435–446, 2019.
[16] Saransh Gupta, et al. Thrifty: training with hyperdimensional computing across flash hierarchy. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.
[17] Behnam Khaleghi, et al. tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 408–413, 2021.
[18] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.
[19] Timothy P. Lillicrap, et al. Random feedback weights support learning in deep neural networks, 2014.
[20] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks, 2016.
[21] Giorgia Dellaferrera et al. Error-driven input modulation: Solving the credit assignment problem without a backward pass, 2023.
[22] Kaiming He, et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.
[23] Zhuowen Zou, et al. Manihd: Efficient hyper-dimensional learning using manifold trainable encoder. *DATE, IEEE*, 2021.