


Hyperdimensional Brain-Inspired Learning for Phoneme Recognition With Large-Scale Inferior Colliculus Neural Activities

Yang Ni , Graduate Student Member, IEEE, Ye Yang, Hanning Chen , Graduate Student Member, IEEE, Xianhui Wang , Nicholas Lesica, Fan-gang Zeng , Fellow, IEEE, and Mohsen Imani , Member, IEEE

Abstract—Objective: Develop a novel and highly efficient framework that decodes Inferior Colliculus (IC) neural activities for phoneme recognition. **Methods:** We propose using Hyperdimensional Computing (HDC) to support an efficient phoneme recognition algorithm, in contrast to widely applied Deep Neural Networks (DNN). The high-dimensional representation and operations in HDC are rooted in human brain functionalities and naturally parallelizable, showing the potential for efficient neural activity analysis. Our proposed method includes a spatial and temporal-aware HDC encoder that effectively captures global and local patterns. As part of our framework, we deploy the lightweight HDC-based algorithm on a highly customizable and flexible hardware platform, i.e., Field Programmable Gate Arrays (FPGA), for optimal algorithm speedup. To evaluate our method, we record IC neural activities on gerbils while playing the sound of different phonemes. **Results:** We compare our proposed method with multiple baseline machine learning algorithms in recognition quality and learning efficiency, across different hardware platforms. The results show that our method generally achieves better classification quality than the best-performing baseline. Compared to the Deep Residual Neural Network (i.e., ResNet), our method shows a speedup up to 74×, 67×, 210× on CPU, GPU, and FPGA respectively. We achieve up to 15% (10%) higher accuracy in consonant (vowel) classification than ResNet. **Conclusion:** By leveraging brain-inspired HDC for IC neural activity encoding and phoneme classification, we achieve orders of magnitude runtime speedup while improving accuracy in various challenging task settings.

Significance: Decoding IC neural activities is an important step to enhance understanding about human auditory system. However, these responses from the central auditory system are noisy and contain high variance, demanding large-scale datasets and iterative model fine-tuning. The proposed HDC-based framework is more scalable and viable for future real-world deployment thanks to its fast training and overall better quality.

Index Terms—Biosignal processing, brain-inspired computing, efficient machine learning, hyperdimensional computing (HDC).

I. INTRODUCTION

IN THE nervous system, information such as sensation, perception, and memory is encoded in neural activities such as action potentials of the neuron population [1]. Identifying how this information is embedded will significantly improve our understanding of the brain mechanism [2], which eventually helps cope with sensorineural disorders and achieve general artificial intelligence. In the past two decades, researchers have been trying to decode useful information from neural activity recordings of various natures. For example, electroencephalography (EEG) and electrocardiography (ECoG) were used in emotion recognition [3], epilepsy detection [4], and imaginary motion recognition [5].

Learning from neural activities and decoding hidden information are never trivial tasks. There are mainly three challenges regardless of which learning mechanism is leveraged. (1) Neural activity recordings usually contain a significant amount of noise, coming from sources like environmental interference, motion artifacts, and biological signals unrelated to target tasks. For most kinds of neural activities, due to the lack of models for simulating clean signals, it is extremely hard to isolate useful signals from a noisy mixture. (2) Neural signals are intrinsically high-dimensional and rich in terms of embedded information, mainly because of the vast number of neurons in brains. Learning with neural activities requires highly sophisticated ML algorithms. (3) In addition, collecting datasets of neural responses on actual subjects is a cumbersome task by itself, thereby limiting the amount of available data for high-quality learning.

Traditionally, learning on neural activities is performed using logistic regression or support vector machine (SVM). More recently, ML algorithms based on deep neural networks (DNN) have become the most popular way to extract information from

Manuscript received 30 November 2023; revised 21 April 2024; accepted 24 May 2024. Date of publication 15 July 2024; date of current version 22 October 2024. This work was supported in part by the DARPA Young Faculty Award, National Science Foundation under Grant 2127780, Grant 2319198, Grant 2321840, Grant 2312517, and Grant 2235472, in part by the Semiconductor Research Corporation (SRC), Office of Naval Research through the Young Investigator Program Award, under Grant N00014-21-1-2225 and Grant N00014-22-1-2067, in part by the Air Force Office of Scientific Research, under Grants FA9550-22-1-0253, and in part by the Wellcome Trust Senior Research Fellowship under Grant 200942/Z/16/Z, and generous gifts from Xilinx and Cisco. (Corresponding author: Mohsen Imani.)

Yang Ni and Hanning Chen are with the Department of Computer Science, University of California Irvine, USA.

Ye Yang, Xianhui Wang, and Fan-gang Zeng are with the School of Medicine, University of California Irvine, USA.

Nicholas Lesica is with the Ear Institute, University College London, U.K.

Mohsen Imani is with the Department of Computer Science, University of California Irvine, Irvine, CA 92697 USA (e-mail: m.imani@uci.edu).

Digital Object Identifier 10.1109/TBME.2024.3408279

EEG and ECoG for classification [6], [7]. However, the power of DNNs comes at a cost: the large number of layers that form their deep structure brings huge computation costs. The bulky backpropagation training process further slows down the model learning process, making DNN very expensive to operate on hardware platforms. In addition, the quality, and more importantly, the quantity of available data for training are dominant factors of DNN learning performance [8]. Therefore, for noisy and expensive-to-collect neural signal data, we need a more efficient ML alternative.

We propose to leverage HyperDimensional Computing (HDC), a lightweight computing framework that mimics the brain at abstract and functional level [9]. HDC uses a high-dimensional representation of the data, i.e., hypervectors, which encode information in a holographic way. This means that the information is evenly distributed in each element of the hypervector. The holographic representation enables hypervector operations (e.g., bundling and binding) that are conceptually similar to neural activities, supporting different types of brain functionalities such as learning, reasoning, and memorization. The benefits of using HDC also include higher learning quality, being more hardware-friendly, and intrinsic computational parallelism. Prior work [10] shows that, by customizing hardware architecture, the HDC-based algorithm achieves orders of magnitude better learning efficiency during real-world deployments.

Prior works have explored possible HDC solutions for bio-signal (including neural signal) processing, achieving significantly better learning efficiency [11], [12], [13]. However, these works have several limitations: (1) the bio-signals they study have relatively small sizes and thus are easier to process like regular feature vectors. (2) the HDC algorithms applied (e.g., N-gram HDC) are not suitable for bio-signals with larger temporal and spatial scales. (3) the heavy use of permutation during encoding and the lack of non-linearity lead to poor efficiency and accuracy when the task scales.

In this paper, we design an HDC-based algorithm for learning on large-scale neural activity, which aims to provide higher efficiency than other major ML algorithms and solve the aforementioned problems in existing HDC algorithms. Our contributions are listed below:

- To the best of our knowledge, for the first time, we leverage HDC for learning on Inferior Colliculus (IC) neural activity. We focus on phoneme recognition since the inferior colliculus is one of the key components in the auditory system. To build the dataset for learning and evaluation, we record IC neural activity from gerbils. In Section II, we introduce more details regarding data collection.
- We design a temporal and spatial-aware HDC encoding method that supports large-scale neural activity inputs such as IC recordings, solving multiple limitations of prior algorithms. During the encoding, we utilize multiple overlapping windows to capture possible shifts in patterns; the inputs are temporally sorted to maintain temporal information while avoiding the use of costly permutation; the encoder also supports the extraction of non-linear features.
- We implement our proposed framework on four different hardware platforms (including embedded GPU

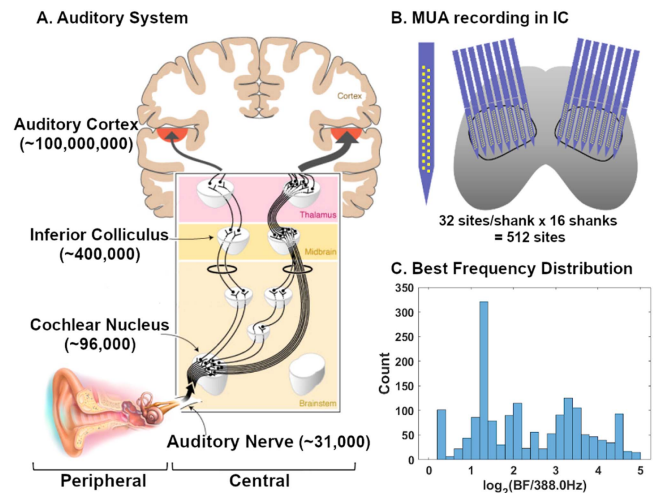


Fig. 1. Overview of: A. Different components of the human auditory system, which can be roughly categorized into peripheral and central auditory system; B. MUA recording in gerbil IC that records 512-channel neural response; C. Channel frequency distribution.

and FPGA) and compare its learning performance and efficiency with four baseline ML algorithms, including multi-layer perceptron (MLP), SVM, and two Convolutional Neural Networks (CNN) (i.e., ResNet and SqueezeNet). The experimental results show that our framework achieves higher or comparable classification accuracy under different settings, compared to the best-performing baseline. In terms of learning efficiency, our HDC-based algorithm is up to $210\times$ faster than Deep Residual Neural Network (ResNet) on a customizable hardware platform (i.e., FPGA). On conventional platforms, we still provide $74\times$ speedup on CPU and $67\times$ speedup on GPU.

II. INFERIOR COLLICULUS NEURAL ACTIVITIES RECORDINGS FOR PHONEME RECOGNITION

Our study focuses on extracting encoded sound information from multi-unit activity (MUA) recordings acquired from the inferior colliculus (IC) of gerbils. Previous studies have looked into phoneme recognition using EEG [14] and ECoG [15], [16]. Compared with EEG and ECoG, MUA recording is able to provide us the more detailed neural activities through spike sorting technique [17]. This allows us to decode information from actual neural activities instead of electric potential fluctuations, which may advance our understanding of neural encoding schemes. The sound information being extracted is 22 consonants or 13 vowels spoken by 10 speakers in English language, which is a harder task than a previous study [18]. All experimental protocols were approved by the U.K. Home Office (PPL P56840C21) on 12/20/2022.

A. Auditory System

The auditory system consists of the peripheral auditory system and the central auditory system (Fig. 1A). The peripheral auditory system converts mechanical sound waves into neural

activities which then progress through the central auditory system consisting of several layers of neurons. The first layer of the central auditory system is the auditory nerve (AN). There are about 31,000 ANs in the human auditory system [19]. Sound waves of different frequencies are converted to neural activities of different ANs, and each AN has its preferred frequency at which it produces the greatest response [20]. After AN, the neural signal propagates through the cochlear nucleus ($\sim 96,000$ neurons [21]), inferior colliculus ($\sim 400,000$ neurons [22]) and finally to the auditory cortex ($\sim 10^8$ neurons [22]) where they further interact with signals from other regions in the cortex to affect sound-related events. In the central auditory system, IC is an important relay that integrates signals from major ascending auditory pathways. This enables the extraction of acoustic information from a relatively small population. Meanwhile, compared with neural activities in the auditory cortex, the activities in IC are less modulated by different other signals, and still predominantly determined by sound waves. Like the auditory nerves, the neurons in IC also have their preferred frequency, i.e., Best Frequency (BF).

B. Neural Activity Recording Techniques

Electrophysiology is the process of directly recording the electric activities of the neurons. This includes EEG, ECoG, MUA recording, and single-unit activity (SUA) recording. From EEG to SUA, the recorded area shrinks while the spatial resolution of the recording increases [23]. For example, in EEG and ECoG recordings, each electrode records the summed electric fluctuation from millions of neurons. On the other hand, MUA records tens or hundreds of neurons through electrode arrays inserted into the brain. Among these methods, MUA recording has significantly better resolution than EEG and ECoG and also covers a larger population of neurons than SUA, keeping a good balance between accuracy and coverage.

C. MUA Recording and Spike Sorting

Due to the invasive nature of the MUA recording, it is done in gerbils, which have a similar frequency response curve to humans [24]. The MUA in use is recorded from multiple gerbils with normal hearing using a 512-channel electrode array (256 sites per hemisphere) at 24414.0625 Hz (Fig. 1B). We select a total of 1669 channels with identifiable spikes across all gerbils for further analysis. The neural activities are down-sampled by a factor of 32 by summing every 32 samples. The final sample rate of the neural time series is about 763 Hz ($24415.0625/32$), which is close to the firing rate of IC neurons [25].

We identify the best frequency (BF) for each of the 1669 channels. These channels are randomly sampled without replacement to form 15 populations according to the BF distribution shown in Fig. 1C. In each population, the number of channels associated with each BF is given by $N(bf) = \left\lfloor \frac{T(bf)}{15} \right\rfloor$, where $T(bf)$ is the total number of channels (out of 1669) with frequency bf . After sampling, there are 100 channels for each population and the excessive channels are dropped. The 15 populations function as multiple trials in our dataset.

D. Phoneme Recognition Using Neural Recordings

During the MUA recording, gerbils were listening to consonant-vowel speech utterances taken from the Articulation Index LSCP (LDC catalog number: LDC2015S12). The selected speech utterances include 286 consonant-vowel combinations (22 consonants and 13 vowels) spoken by 10 speakers (5 males and 5 females). To further test if the information can be extracted from noisy speech. The speech utterances were mixed with two additive speech bubble noises at 0 dB SNR and presented to gerbils during MUA recording. For each condition (clean, noisy_1, noisy_2), the speech utterances were presented randomly at an intensity of 60 dB SPL and 85 dB SPL. The first 188 time bins (~ 246 ms) of the neural activities after utterance onset were considered as the responses to the speech utterances. All classification tasks were performed based on the neural activities of 100 channels by 188 time bins. Our dataset contains 42900 such multi-channel neural activities (286 consonant-vowel, 10 speakers, and 15 populations), forming a large-scale neural activity dataset.

III. HYPERDIMENSIONAL COMPUTING: MOTIVATION AND BACKGROUND

As is shown in the previous section, human brains integrate and process external information, such as sound, with a large number of neurons. Inside human brains, the cerebellum cortex is considered a significant component in brain cognitive activities [26]. The information of sensory inputs is stored in the cerebellum cortex using high-dimensional neural activity patterns. Experimental evidence shows that it actively participates in semantic processing as well as phonological processing [27]. This motivates us to leverage HDC for neural activity processing and phoneme recognition. HDC is backed by a set of hypervector operations that tries to simulate brain functionalities such as short-term memorization and learning through high-dimensional patterns. In the rest of this section, we introduce some of the major HDC operations, and how they help achieve brain-style learning and memorization.

In HDC, randomness is the key to enabling fast learning, robustness, and other favorable characteristics [9]. To begin with, we generate random bipolar hypervectors \vec{H} where each element h is identically and independently distributed (i.i.d.), e.g., $\vec{H} \in [+1, -1]^D$ where $P(h = 1) = P(h = -1) = 0.5$. When the dimensionality of hypervectors is very large, e.g., $D \simeq 10000$, randomly generated hypervectors are pseudo-orthogonal to each other. For instance, we check the cosine similarity between \vec{H}_i and \vec{H}_j as the following: $\delta(\vec{H}_i, \vec{H}_j) \approx 0$ for every $i \neq j$. In fact, there is a high probability that two hypervectors randomly picked from the high-dimensional space have nearly zero cosine similarity. It should be noted that bipolar or binary hypervector is used for simplicity here, however, hypervectors with higher precision such as real/complex value also possess this property. For human beings who are more familiar with lower-dimensional space, pseudo-orthogonality is one of the non-intuitive properties of hypervector space, and there are a few more that help define HDC operations. The following are

major HDC operations that form the so-called ‘Multiply-Add-Permute’ in HDC [28]:

- **Binding** ($*$) stands for element-wise multiplication. The binding of two orthogonal hypervectors will result in a third hypervector that is near-orthogonal to both constituents. In the case of $\vec{H}_k = \vec{H}_i * \vec{H}_j$, the dot product between \vec{H}_k and \vec{H}_i (or between \vec{H}_k and \vec{H}_j) will be very close to zero as their elements all follow a symmetric random distribution. And if we compute the cosine similarity δ between these two pairs of hypervectors, we observe that $\delta(\vec{H}_i, \vec{H}_i * \vec{H}_j) \approx \delta(\vec{H}_j, \vec{H}_i * \vec{H}_j) \approx 0$. The near-zero cosine similarity also shows that these hypervectors are near-orthogonal to one another.
- **Bundling** ($+$) is defined as the element-wise addition: $\vec{H}_{ij} = \vec{H}_i + \vec{H}_j$. Unlike binding operation that creates dissimilar hypervectors, the output of the bundling operation preserves similarity to inputs. In terms of cosine similarity, this means $\delta(\vec{H}_{ij}, \vec{H}_i) \gg 0$ and $\delta(\vec{H}_{ij}, \vec{H}_j) \gg 0$. This operation reveals that HDC is capable of memorizing information by bundling different hypervectors and checking existence of queries via similarity metrics.
- **Permutation** (ρ) rotate-shifts hypervectors by one element. It creates pseudo-orthogonal hypervectors based on existing ones, i.e., $\delta(\vec{H}_i, \rho\vec{H}_i) \approx 0$. One important function is to preserve temporal relations between different inputs. With permutation, we can distinguish between two different sequences with the same components, e.g., $\vec{A} = \vec{H}_i + \rho\vec{H}_j + \rho\rho\vec{H}_k$ and $\vec{B} = \vec{H}_j + \rho\vec{H}_i + \rho\rho\vec{H}_k$.

IV. AN HDC-BASED FRAMEWORK FOR LEARNING ON NEURAL ACTIVITY

In this paper, we propose a hyperdimensional learning algorithm for phoneme recognition, which operates on the IC neural activity dataset introduced in Section II. In the next few subsections, we first introduce our HDC encoder designed for large-scale neural signals (Section IV-A). Then we go through the HDC classifier with adaptive model update (Section IV-B).

A. HDC Encoder for Neural Activity Recordings

1) Challenges in Existing HDC Neural Signal Encoding:

Current HDC encoding methods have multiple limitations including quantization loss and the lack of non-linearity. Recent HDC encoding methods designed for bio-signals like EEG and EMG [11], [29] only focus on bipolar hypervectors (i.e., the element being either +1 or -1), which lack the capacity to capture complex patterns. For example, encoding real-value inputs is challenging because the encoder needs to first quantize them to a finite number of levels. This step already brings noise to the learning process and thus quality loss. The most significant drawback, however, is that bipolar hypervectors with bundling and binding (as we introduced in Section III) fail to consider correlations between inputs and also within each input vector.

The task defined in this paper utilizes large-scale neural recordings that are rich in both temporal and channel-wise patterns. In terms of the time-series length for each sample,

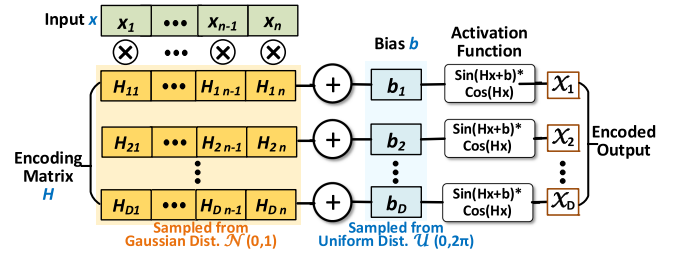


Fig. 2. Overview of the Kernel-based HDC Encoding composed of an encoding matrix, bias hypervector, and activation function; it maps inputs from original space to hyperspace.

the IC neural recordings in our task are significantly longer than previous workloads in [11], [12]. In addition, our MUA neural recordings have 100 channels, which is much higher compared with other healthcare tasks. For example, ECG sensors for arrhythmia monitoring and emotion detection tasks only have 2 channels [30], [31].

Existing HDC-based solutions [11], [12] heavily rely on aggressive downsampling to reduce the length of the bio-signal samples because large-scale inputs are not well-supported. Even after downsampling, they only focus on smaller clips within the long time series for classification, which limits the ability to extract global patterns. The extensive use of permutation for maintaining temporal sequence also adds significant learning overhead when the task scales.

2) Kernel-Based Non-Linear Encoding: As a solution to these problems, we utilize a kernel-based HDC encoding method with non-linearity [32]. It gets inspired by the Radial Basis Function, i.e., RBF kernel, which is defined as $K(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}}$, where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and σ is the length scale parameter. In addition, prior works propose the kernel trick [33], [34] that approximates shift-invariant kernels like the RBF kernel using dot-products of another mapping \mathbf{z} : $K(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \approx \mathbf{z}(\mathbf{x}) \cdot \mathbf{z}(\mathbf{y})$, where Φ is the implicit mapping corresponding to the RBF kernel. A suitable and high-dimensional mapping \mathbf{z} will lead to dot products that converge to the target kernel function in expectation.

We leverage this trick and construct an HDC encoder by defining a high-dimensional mapping $\mathbf{Z}(x)$, following the results presented in [33]:

$$\text{Hypervector } \vec{\mathcal{X}} = \mathbf{Z}(\mathbf{x}) = \cos(\mathbf{H}\mathbf{x} + \mathbf{b}) * \sin(\mathbf{H}\mathbf{x})$$

As shown in Fig. 2, for input features of size n , \mathbf{H} is a $D \times n$ matrix with its elements randomly sampled from Gaussian distribution $\mathcal{N}(0, 1)$ and $\mathbf{b} \in \mathbb{R}^D$ is a bias term generated using $\mathcal{U}(0, 2\pi)$. Therefore, this encoder will map inputs $\mathbf{x} \in \mathbb{R}^n$ to hypervectors with a significantly larger dimensionality D ($D \gg n$).

The connection shown between HDC and kernels has more profound meanings. A kernel-based HDC encoder has multiple advantages, with the most immediate one being avoiding the quantization loss during encoding. Notice that the encoding matrix \mathbf{H} maps the input directly to hypervector space without the need to discretize inputs and generate so-called ‘Level Hypervectors’. The second benefit is that, by introducing kernels

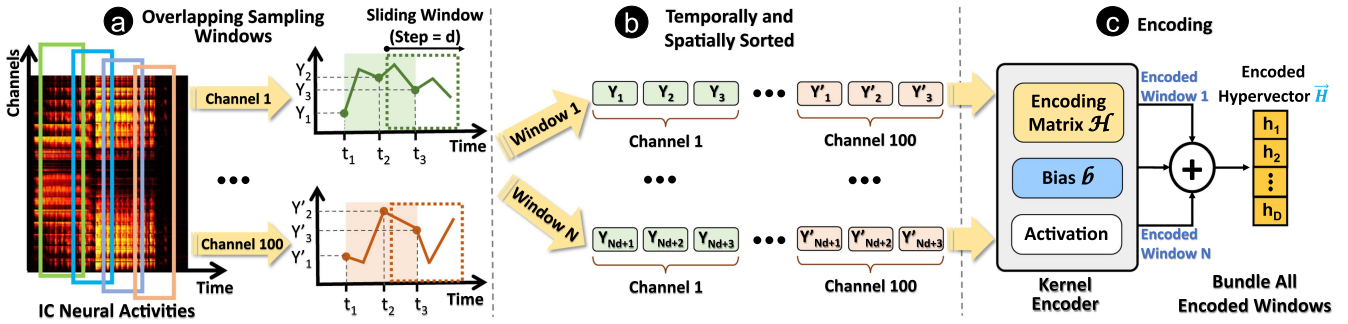


Fig. 3. Spatial and temporal-aware HDC encoding for IC neural activity recordings: (a) Use a sliding window to cover a long time series, (b) within the same window, concatenate values from all channels, (c) bundle encoded hypervectors of different window to create the final representation.

to HDC, encoded hypervectors retain a good space similarity. In other words, the similarity between two hypervectors approximates the corresponding kernel value, which is shift-invariant and only depends on the distance in the original space. The third benefit comes from the RBF kernel itself. As a universal kernel [35], the RBF kernel guarantees an optimal approximation to arbitrary bounded and continuous functions.

3) Spatial and Temporal-Aware Encoding: The previous section defines a good HDC encoding method, however, it is not yet suitable for large-scale neural activities encoding. To bridge this final gap, we need to consider both spatial and temporal information during HDC encoding.

For illustration, we use a single IC recording that corresponds to a consonant-vowel pair, as shown in Fig. 3. Recall Section II, each neural activities recording contains 100 channels, and the length is 188. One naive way to handle this 2D input in prior HDC solutions is to flatten it as a 1D vector with 18800 elements, similar to traditional ML algorithms such as MLP and SVM. Although direct flattening to 1D is compatible with the HDC kernel-based encoder, it comes at a cost. In neural activity analysis, the biggest difficulty is that the locations of key patterns shift from sample to sample and they may reside in multiple smaller clips. Therefore, if the encoding process considers the whole flattened input, HDC loses the ability to detect the shift of patterns. In the case of phoneme recognition, the onset of consonants may shift due to different speakers and vowels.

To address this problem, before encoding, we process the neural activity recording through a series of overlapping windows. In Fig. 3(a), we use multiple sliding windows to effectively cover the whole recording. For simplicity, the window size is set to 3 in the figure and we only show the first and the last channel. We also assume a step size of d and N overlapping windows. In each channel, a window includes 3 neural activity values $\{Y_1, Y_2, Y_3\}$ that correspond to the time points $\{t_1, t_2, t_3\}$. To encode these values with temporal information, we maintain the order of inputs before mapping to hypervectors. As shown in Fig. 3(b), for each window, we select values in the first channel in the order of time and then continue with the next channel. This results in a longer vector of 300 elements since there are 100 channels in each neural signal recording. This sorting makes sure that the encoder is aware of the temporal and spatial location of each element. The sorted vector for each window (i.e., N vectors in total) is then ready for encoding with our kernel-based encoder.

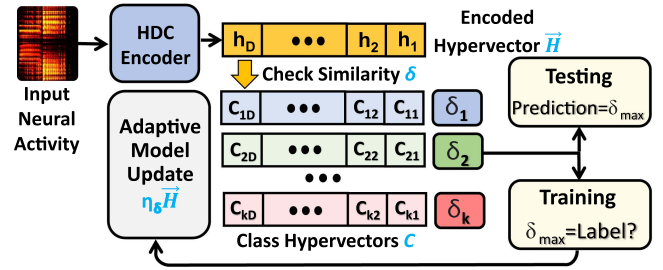


Fig. 4. HDC inference and adaptive model update that are based on lightweight hypervector similarity check.

In Fig. 3(c), we use the bundling operation to combine encoded hypervectors for different sampling windows and obtain the final output as a single hypervector. Bundling ensures that the output hypervector contains information from every window.

By encoding inputs using multiple overlapping windows, our encoder can effectively capture shifted patterns due to speaker and phoneme variations. Suppose that one neural activity pattern that showed up in training is delayed in testing samples. Our encoder ensures that they remain similar because these encoded hypervectors contain a large number of similar terms during the bundling of sliding windows.

Finally, our encoder design also has an efficiency benefit. Specifically, our encoder is free of permutation operation, which is inefficient on CPU platforms, not to mention low-power devices. In our encoder, we preserve the temporal and spatial order before mapping to hyperdimensional space and thus, the computation overhead is low. On the other hand, prior works require thousands of permutation operations to mark temporal information for the whole time-series.

B. HDC Classifier With Adaptive Model Update

HDC learning, compared with widely adopted DNN, has higher efficiency and faster model training. The reason is that HDC learning directly operates over encoded hypervectors. Although the dimensionality is higher than usual, the HDC operations are lightweight and easy to parallelize. As shown in Fig. 4, the learning process is free of high-cost computations such as backpropagation. For both HDC training and testing phases, the input neural activity is first mapped to a single encoded hypervector \vec{h} using the encoder design we introduced

in Section IV-A. In inference, we check the cosine similarity between the encoded query hypervector $\vec{\mathcal{H}}$ and every class hypervector \vec{C}_k with: $\delta(\vec{\mathcal{H}}, \vec{C}_k) = \vec{\mathcal{H}} \cdot \vec{C}_k / (|\vec{\mathcal{H}}| |\vec{C}_k|)$. The model predicts the label of this query as $\text{argmax}_k(\delta(\vec{\mathcal{H}}, \vec{C}_k))$. The HDC training process, thanks to the memorization capability of HDC, can be as simple as bundling hypervectors of the same class [9]. However, the major drawback of naive hypervector accumulation is that class hypervectors are quickly saturated by common patterns, while a few of the training samples with uncommon patterns are not well-learned by the model.

In this paper, we apply the adaptive model update during the HDC training. In Fig. 4, we show that the update is now adaptive based on the similarity to each class hypervector. For example, if the pattern is already memorized quite well in the hypervector, i.e., similar samples are met before, the update of the class hypervector should be minimal to avoid hypervector saturation. On the other hand, if the pattern does not exist in the correct class hypervector, then the update should be slightly more aggressive. Therefore, a large number of common patterns will no longer dominate the class, and uncommon patterns will still be effectively learned by the model.

In detail, we define the similarity-based adaptive learning rate η_δ . For an input with label k , and if the HDC model predicts incorrectly to class k' , we need to update model hypervectors for both classes. To begin with, the update procedure for the class hypervector \vec{C}_k is:

$$\vec{C}_k = \vec{C}_k + \eta_\delta \times \vec{\mathcal{H}} = \vec{C}_k + \eta \times (1 - \delta_k) \times \vec{\mathcal{H}}$$

where η is a base learning rate and δ_k is short for the similarity $\delta(\vec{\mathcal{H}}, \vec{C}_k)$. δ_k indicates to what extent the input pattern already exists in \vec{C}_k . Since the maximum value of cosine similarity is 1, the adaptive learning rate $\eta \times (1 - \delta_k)$ limits the added patterns to eliminate model saturation. As for the class hypervector $\vec{C}_{k'}$:

$$\vec{C}_{k'} = \vec{C}_{k'} + \eta_\delta \times \vec{\mathcal{H}} = \vec{C}_{k'} + \eta \times (\delta_{k'} - \delta_k) \times \vec{\mathcal{H}}$$

where $\delta_{k'} - \delta_k$ represents to what extent the model should be updated. In the case when $\delta_{k'} - \delta_k \gg 0$, it shows that a large mismatch happens and the corresponding learning rate should be larger. Meanwhile, a smaller value means that only a slight update is needed for the correct classification. This training process is also iterative, where the HDC model becomes more accurate after multiple epochs of training.

C. FPGA Accelerator Design

For better efficiency in real-world applications, HDC-based algorithms can be accelerated on various hardware platforms. In this paper, we focus on the FPGA as HDC models are well-suited for customized domain-specific accelerators. HDC models are characterized by natural computing parallelism and low element precision which has been shown in many previous study [10], [36], [37], [38]. In contrast, CPUs and GPUs are not ideal for accelerating HDC models [36]. CPUs lack sufficient computing parallelism, while GPUs incur high data communication overhead between the CPU and GPU. Additionally, GPUs have relatively high power consumption due to their support for high-precision floating-point operations.

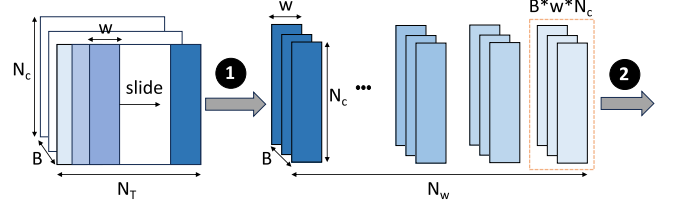


Fig. 5. Data offloading scheduling on CPU side: (1) Process the raw data with sliding window on CPU, (2) data batches are then offloaded onto FPGA.

In particular, we designed a CPU-FPGA heterogeneous acceleration framework. In Fig. 5, we present the data offloading scheduling process on the CPU side. Here we suppose the batch size is B , the time series length is N_T , the total number of channels is N_c , and the sliding window size is W . By covering the whole time series with overlapping windows (Fig. 5①), we will generate total N_w data windows at the CPU side. Suppose the sliding window step size is Δ , The size of N_w is gonna be:

$$N_w = \left\lfloor \frac{N_T - W}{\Delta} \right\rfloor + 1$$

The shape of each data window is $B \times W \times N_c$. The host program will offload one data window into the kernel FPGA accelerator (Fig. 5②) during both training and inference stages. The communication interface between the host CPU and kernel FPGA is PCIe [39].

In Fig. 6(a) we present the FPGA kernel architecture design. The host CPU will offload the input data into the kernel FPGA via PCIe link and Xilinx AXI Direct Memory Access (DMA) IP (Fig. 6①). To maximize the FPGA on-chip resource utilization rate, we conduct two-dimensional data partitioning. This means that we partition both input data and HDC encoder during hypervector encoding and similarity search. Specifically at Fig. 6②, we reshape each input data from the original channel size (N_c) into chunk size T . The reason here is that a large systolic array leads to synthesis and routing challenges, considering the limited resources and area of FPGA [40]. Instead, as is shown in Fig. 6(b), we choose to deploy multiple smaller-sized systolic array IPs such that the total PEs are the same. Our main goal is to maximize the speedup by maximizing the data computing parallelism. Therefore, we design specific datapaths and re-architect the datapath, such that our design can process as many data channels as possible on limited on-chip processing elements. Each systolic array IPs have multiple computing unit (CU) IP. In Fig. 6(c), we present the micro-architecture design of CU IP. Our design is an output-stationary systolic array; inputs to each CU will be passed to other neighboring CUs downstream; and the register in each CU accumulates the multiplication results and outputs the final value (i.e., output stays in each CU). We use a first in first out (FIFO) buffer to store each chunk of input data (Fig. 6③). Here we use Y_i to represent the i^{th} input data chunks. The dimension of each buffer-stored input data chunk is $B \times T$ and there are total S entries inside the buffer. The size of S is:

$$S = \frac{N_c \times W}{T}$$

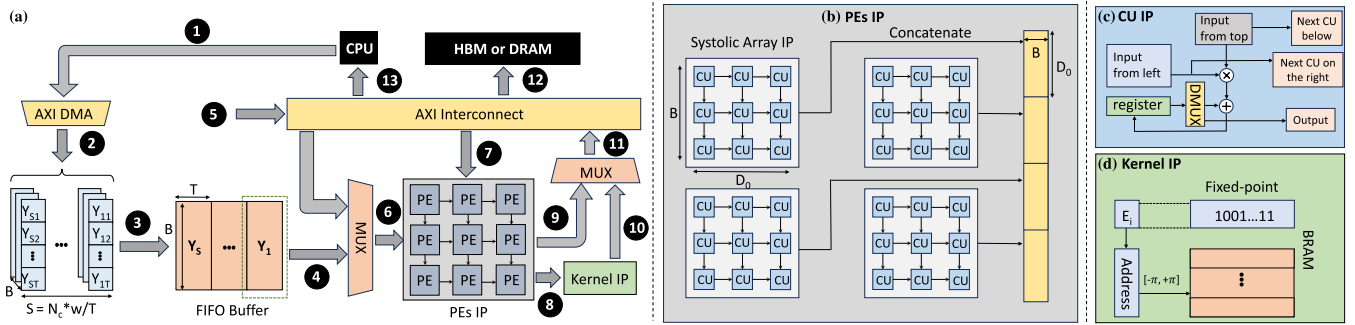


Fig. 6. (a) An overview of our proposed FPGA acceleration framework for HDC-based phoneme recognition. (b) Processing Elements (PE) IP architecture: 4 small systolic array computing in parallel for hypervector operations. (c) Systolic array computing unit (CU) IP microarchitecture. (d) Kernel function IP architecture for efficient computation of trigonometric functions.

We leverage Processing Elements (PEs) IP to accelerate the matrix-to-matrix multiplication (M2MM) which is the foundation of hypervector operation. For base hypervector and input data multiplication acceleration, we load the input data window from the FIFO buffer (Fig. 6(4)) into PEs (Fig. 6(6)) and for HDC similarity search, we load the encoded hypervector from on-chip storage into the PEs IP via Xilinx AXI Interconnect IP. Notice that depending on the type of FPGA board, the on-chip storage could be either DRAM or high bandwidth memory (HBM). The latter one provides faster data movement thanks to its multiple parallel channels that form high bandwidth. We include a multiplexer IP controlled by the CPU input signal (Fig. 6(5)) to decide the data path that PEs IP takes. The input data of PEs IP could be either unencoded raw input data or encoded hypervector. We also pre-store HDC encoder hypervectors and class hypervectors inside on-chip storage (Fig. 6(7)). As we mentioned above, to decrease FPGA synthesis difficulty, we choose to implement multiple small-sized systolic arrays IP inside the PEs IP. Each small size systolic array IP's dimension is $B \times D_0$ where B is the training and inference batch size and D_0 is the hypervector chunks size [41]. This means each systolic array has $B \times D_0$ computing unit (CU) IP. Each small systolic array IP's output vectors will be concatenated together.

After the M2MM acceleration, the output vectors will be passed into the activation kernel function IP during the encoding stage (Fig. 6(8)) or stored back into on-chip storage during the similarity search stage (Fig. 6(9)). To save the on-chip resource usage (especially for DSP usage), we choose to use on-chip memory (such as BRAM) as codebook triangular function table [10], [42], [43]. As is shown in Fig. 6(c), we treat each hypervector element (fixed-point number) as the input address. Then we pre-store all possible trigonometric function results inside on-chip memory such as BRAM. Notice that for trigonometric functions (such as sine and cosine), valid inputs range from $-\pi$ to π , and thus we don't need to use the full data precision as the input address. Instead, we quantize the hypervector elements into $-\pi$ to π and only use valid bits as input address.

After the kernel function IP (Fig. 6(10)), we store the encoded hypervector into the on-chip storage (Fig. 6(12)) via Xilinx AXI Interconnect IP (Fig. 6(11)). During the HDC model training, we only perform encoding operation once and reuse pre-encoded hypervector to train the class hypervectors [44]. This customized

datapath design significantly reduces unnecessary encoding computation and CPU-FPGA communication overhead. During the inference, we will pass prediction results back to the host CPU (Fig. 6(13)) via Xilinx AXI DMA IP.

V. EXPERIMENTAL RESULT

A. Experiment Settings

We evaluate our proposed HDC-based neural activity learning framework on a diversity of hardware platforms, including AMD Ryzen 5 3600X CPU, NVIDIA RTX 3070 GPU, Jetson Orin embedded GPU, and Xilinx Alveo U50 FPGA. Jetson Orin has been widely applied to applications such as robotics and self-driving, where power consumption needs to be minimized. Xilinx Alveo U50 is also a low-power FPGA targeting embedded usage. Using these four platforms, we try to cover a broader range of hardware environments for real-world deployments. On CPU and GPU platforms, we utilize Pytorch for implementing our algorithm; and we use Vitis for the deployment on FPGA. As mentioned in the task introduction (Section II), the dataset we focus on has six different configurations, i.e., 3 noise settings * 2 speech intensity levels. In the following sections, we provide accuracy and runtime results for these datasets with different target tasks, including consonant classification and vowel classification. For these two classification tasks, by default, we only use neural activities that are related to either consonants or vowels. That is, for consonant classification, we zero-mask most of the neural responses for vowel sounds, and vice-versa. The coarse boundary between vowels and consonants is identified during the collection of the dataset, which is visible on recordings. However, we also try to classify using non-masked inputs, and the results are shown in Section V-F.

Unless otherwise specified, we generate the training and testing datasets using a 'speaker-dependent' fashion with the 80-20 split. In other words, the split happens within 130 vowel-speaker pairs for the consonant classification and 220 consonant-speaker pairs for the vowel classification. It is speaker-dependent because the speech utterances of a certain speaker may show up in both training and testing samples, although with different vowels or consonants. In Section V-D, we also consider the 'speaker-independent' situation, which is intuitively harder due

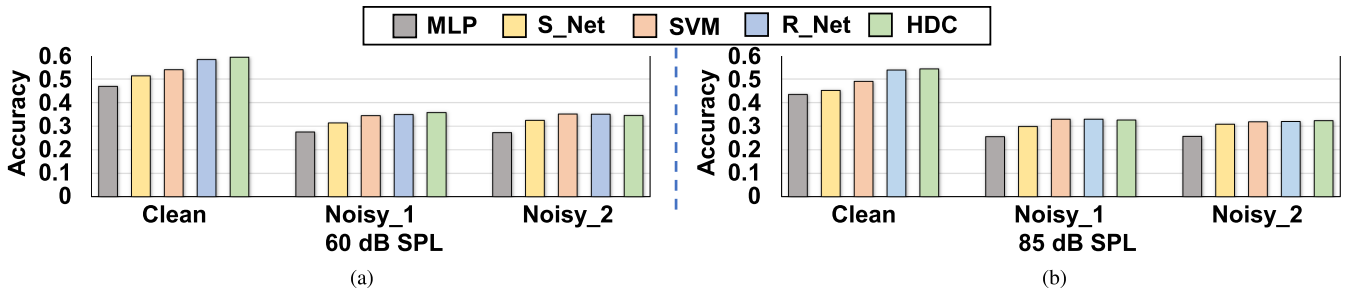


Fig. 7. Consonant classification accuracy comparison between our HDC-based algorithm and four baselines. The accuracy results are reported across six datasets of different noise and speech intensity conditions (in (a) we use a speech intensity of 60 dB SPL, and (b) has an intensity of 85 dB SPL). In this figure and the rest, we use ‘S_Net’ as short for SqueezeNet and ‘R_Net’ for ResNet.

TABLE I

THE RESOURCE UTILIZATION OF HDC MODEL ACCELERATION ON XILINX ALVEO U50. HERE FPGA KERNEL FREQUENCY IS 200 MHz AND THE POWER CONSUMPTION IS 23 W

	LUT	FF	DSP	BRAM	URAM
Total	685.1K	402.1K	7	768	32
Available	872K	1743K	5952	1344	640
Utilization	78.5%	23.1%	~0%	57.1%	2.5%
$T_{training} = 5.40s, T_{inference} = 0.26s$					

to the shift of sample distribution between training and testing datasets.

To illustrate the advantage of using HDC in this task, we compare its results with other ML algorithms in terms of learning efficiency and classification accuracy. We select four different ML algorithms that range from conventional ones to state-of-the-art. The first one is a simple MLP with two hidden layers (each has 512 and 128 neurons). The second one is SVM with RBF kernel which is frequently used in bio-signal classification tasks. In experiments, the multi-channel neural activity is flattened before learning with MLP and SVM. The last two are deep CNNs. In our experiments, we choose SqueezeNet [45] and Resnet18 [46]. Both networks are loaded directly from PyTorch and trained from scratch. Specifically, we use SqueezeNet version 1_1 which aims at lower computation cost without sacrificing the quality. We collect GPU results for CNNs and HDC using PyTorch and the ones for SVM through ThunderSVM [47]. We accelerate the HDC model on Xilinx Alveo U50 based on previous HDC accelerator design [10]. In Table I, we present the resource utilization of our HDC acceleration. LUT is short for look-up table, FF is Flip-Flop, DSP stands for digital signal processing unit, BRAM means block RAM that supports memory-intensive operations, and URAM is ultra RAM with larger capacity and optimized for high bandwidth. We also report other model’s performance (SVM, MLP, S_Net, and R_Net) on Alveo U50 based on previous works [48], [49]. The training runtime reported in this section follows the iterative learning setting, both MLP and CNNs are fully trained with 100 epochs, and 150 epochs for HDC. In Section IV-A, we show that our HDC-based algorithm has three hyperparameters including the dimensionality D , the window size, and the step size between neighboring windows. As for the default settings in the following

experiments, we set the window size to 40, the step size to 5, and $D = 10000$.

B. Consonant Classification Accuracy & Efficiency Comparison

In this section, we present the results of our HDC-based framework solving the consonant classification on all six datasets, i.e., with different noise and speech intensity settings. Our experimental results show that the HDC-based algorithm achieves better or comparable results, compared to the best-performing baseline algorithms, i.e., ResNet. As shown in Fig. 7(a), when the speech intensity is set to 60 dB SPL, HDC obtains nearly 60% accuracy. On the other hand, MLP and SqueezeNet do not perform well in these datasets. In Fig. 7(b), we provide results for datasets with 85 dB SPL speech intensity, where the accuracy of HDC is still comparable to ResNet and outperforms other baseline algorithms. By comparing the results of 60 dB SPL and 85 dB SPL, we notice an interesting phenomenon: the classification accuracy decreases while the speech intensity increases across all noise conditions and algorithms. This may be due to the saturation of IC neurons since their firing rates have an upper limit. The high-intensity sound (85 dB SPL) caused more neurons to fire at their maximum rate which makes the IC firing pattern less differentiable. For the rest of the experimental results, we only focus on datasets of 60 dB SPL due to the limited space and similar performance trend in 85 dB SPL datasets. In addition, the normal human voice intensity is about 60 dB SPL.

We also compare the total training runtime for all five classification algorithms on CPU, GPU, and FPGA platforms. Results reported here are averaged across different dataset configurations, although the runtime for each is very similar. In Fig. 8(a), we observe that HDC requires significantly shorter training time on the CPU, compared to all four baseline algorithms. Specifically, our HDC-based classification framework requires only about 550 seconds, i.e., around $74\times$ faster than ResNet. In Fig. 8(b) and (c), we report the runtime collected on the embedded Jetson GPU and more powerful RTX 3070. The speedup brought by HDC is about $62\times$ and $67\times$ respectively, compared to ResNet running on the same hardware. In addition, we find that the HDC training time on the CPU is still significantly faster, even compared with the GPU runtime of CNN and

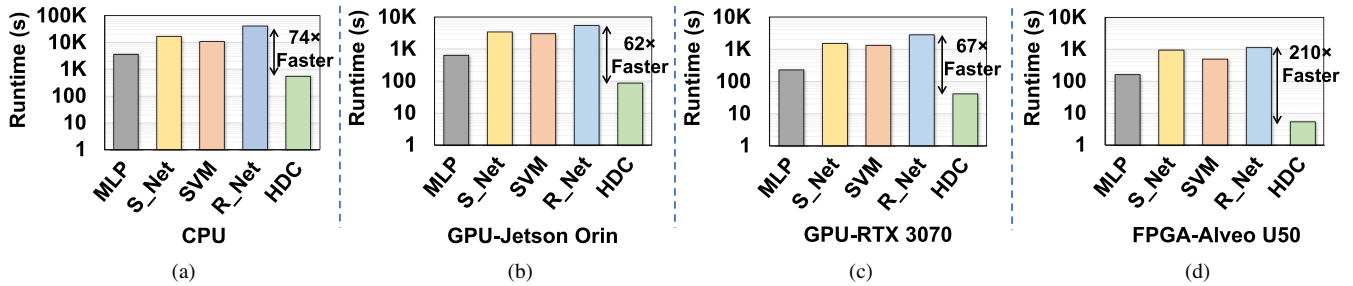


Fig. 8. Consonant classification efficiency comparison between our HDC-based algorithm and four baselines with deployments on 4 different hardware platforms: (a) CPU, (b) an embedded GPU-NVIDIA Jetson Orin, (c) a desktop GPU-NVIDIA RTX 3070, (d) Alveo U50 FPGA.

SVM algorithms. Fig. 8(d) presents the learning runtime of each algorithm on the Alveo U50.

Due to the great design flexibility, FPGA is able to further accelerate all five algorithms with faster learning runtime. However, our algorithm based on HDC benefits more from the customizable features of FPGA and shows a significant $210\times$ speedup compared to ResNet. The learning runtime is reduced from more than 18 minutes to just 5 seconds. HDC models with natural computing parallelism and holographic representation with better tolerance against low element precision, show more benefits during FPGA acceleration. With our architecture design introduced in Section IV-C, HDC-based algorithms achieve higher computation parallelism and more efficient usage of limited resources than all other baseline algorithms. As shown in Table I, the hypervector computation exploits FPGA Look-up tables (LUTs) that are significantly more efficient than Digital Signal Processors (DSPs). In addition, by customizing the computation datapath, we can reuse previous computation results and pipeline the encoding and classification operations.

Therefore, our proposed HDC-based solution is a highly efficient and more effective algorithm compared to CNN/SVM when learning noisy neural activity data. In this paper, we mainly illustrate two advantages of HDC-based algorithms including better classification quality and higher training efficiency. As shown in Table I, the average inference runtime of HDC on FPGA is 0.26 seconds, about $5\times$ faster than ResNet (1.3 seconds). However, this is negligible when compared to the learning runtime and thus we will only focus on training in the rest of the sections.

C. Hyper-Parameter Tuning

In this section, we explore the effect of varying two HDC hyperparameters on CPU training time and consonant classification accuracy, including the window size and the step size. Based on the results, we select the default setting for these two hyperparameters. In Fig. 9(a), we vary the window size from 20 to 70. When the window size increases, although the total number of overlapping windows decreases due to the fixed step size and length of neural recordings, the input size of the HDC encoder increases; and this leads to the training runtime trend shown in this figure. Compared to the baseline ResNet accuracy, HDC achieves better or at least comparable accuracy. We select 40 as our default window size setting for its lower

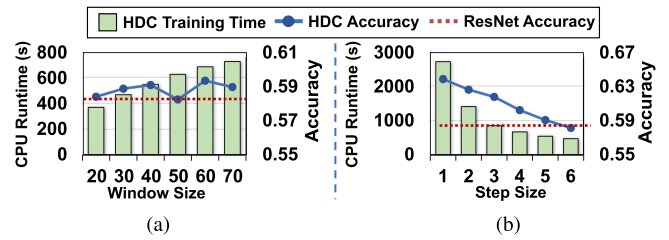


Fig. 9. HDC consonant classification performance with different window sizes and step sizes. The results shown are for the 60 dB SPL clean neural activity dataset. In (a), the step size is 5; in (b), the window size is 40; the dimensionality is 10 k for both.

training time and satisfying accuracy. Then we fix the window size at 40 and explore the effect of step size on learning quality and efficiency. In Fig. 9(b), we change the step size from 1 to 6 and observe that both the training runtime and classification accuracy increase when the step size decreases. When the step size is smaller or equal to 5, the classification accuracy of HDC is higher than ResNet. Therefore, we choose 5 as the default step size for a shorter training runtime. Notice that the consonant classification accuracy of HDC is able to achieve nearly 64% with a training runtime of fewer than 3000 seconds on CPU. The maximum HDC accuracy shown here is significantly higher than all baseline algorithms, yet it is still much faster than CNN and SVM.

D. Speaker Independent Consonant Classification

With the default speaker-dependent setting (recall Section V-A), the split of the training and testing datasets only considers separating different pairs of speakers and vowels for consonant classification. That is, speech from the same speaker can show up in both training and testing datasets. In this section, we generate datasets that follow the speaker-independent setting. In other words, we will randomly select two out of ten speakers and use the neural recordings corresponding to their speech as the testing dataset. The rest eight speakers will count toward the training dataset. This leads to a significant train-test gap and thus becomes more challenging for all machine learning algorithms.

In Table II, we present the consonant classification results and the training runtime for different algorithms and hardware platforms. Our HDC-based algorithm outperforms all baselines in both accuracy and learning efficiency. For example, compared

TABLE II
CONSONANT CLASSIFICATION PERFORMANCE COMPARISON UNDER THE SPEAKER-INDEPENDENT SETTING ACROSS ALL 60 dB SPL DATASETS

	Clean Accuracy	Noisy_1 Accuracy	Noisy_2 Accuracy	GPU _{Jetson} Runtime (s)	GPU _{RTX} Runtime (s)	FPGA _{Alveo} Runtime (s)
MLP	0.400	0.285	0.203	664.3	314.4	221.4
S_Net	0.403 (+0.003)	0.274 (-0.011)	0.272 (+0.069)	3335.2	1613.4	1014.5
SVM	0.508 (+0.108)	0.298 (+0.013)	0.307 (+0.104)	3521.4	1757.5	655.6
R_Net	0.403 (+0.003)	0.244 (-0.041)	0.265 (+0.062)	5298.7	2924.2	1193.6
HDC	0.551 (+0.151)	0.307 (+0.022)	0.324 (+0.121)	85.8	41.7	5.6

The bold text stands for the best-performing model (most accurate & least runtime).

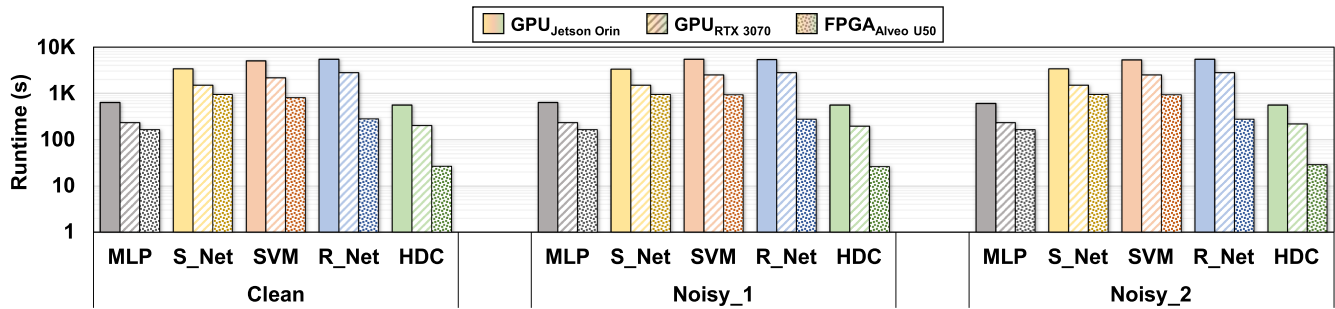


Fig. 10. Vowel classification training runtime comparison with different baseline algorithms and across hardware platforms: experiments are carried out on datasets with three different noise conditions; HDC-based algorithm constantly achieves better efficiency.

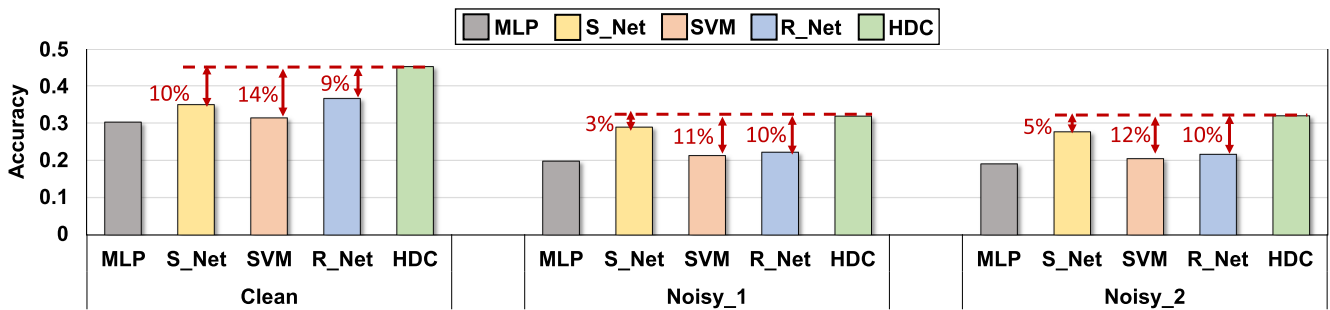


Fig. 11. Vowel classification accuracy comparison across different noise conditions: for all three settings (Clean, Noisy_1, and Noisy_2), HDC-based algorithm constantly outperform the rest with up to 14% improvements.

to SVM (ResNet), results for the clean dataset show that HDC achieves about 4% (15%) higher accuracy; in terms of runtime, HDC is about 42 \times (70 \times) faster on RTX 3070 GPU; as for FPGA, the speedup is 117 \times (213 \times). To conclude, the fact that HDC obtains higher accuracy shows that it may generalize better than other baselines in the case of dataset shift.

E. Vowel Classification Accuracy & Performance Comparison

In this section, we focus on vowel classification using IC neural recordings. We test different algorithms on 60 dB SPL datasets of three different noise conditions. In Fig. 10, we present the learning runtime comparison. We run a simple HDC hyperparameter tuning similar to the one described in Section V-C and set the step size to 2 and the window size to 80. Across all hardware platforms, the HDC-based algorithm achieves significantly faster training than baselines such as SqueezeNet, ResNet, and SVM. For example, compared to the best-performing SqueezeNet that is efficiency-orientated, our

HDC provides significantly higher classification accuracy and is on average 35 \times faster in training on FPGA.

In Fig. 11, we observe that our proposed HDC algorithm achieves the highest accuracy as well as the lowest learning runtime in all three datasets and across different hardware platforms. HDC-based method provides an average accuracy improvement of 9.7%, 12.3%, and 6% compared to ResNet, SVM, and SqueezeNet, respectively. One finding to note is that, across all algorithms, the vowel classification accuracy is lower than the one for consonant classification, even though there are fewer vowels than consonants. This is due to the larger variability of vowel pronunciations across different speakers and different words than consonant pronunciations [50].

F. Classification on Non-Masked Datasets

For the previous sections, we utilize the available datasets with either consonant or vowel masked. However, we also test different algorithms using datasets without masks, and we present both consonant and vowel classification results using the

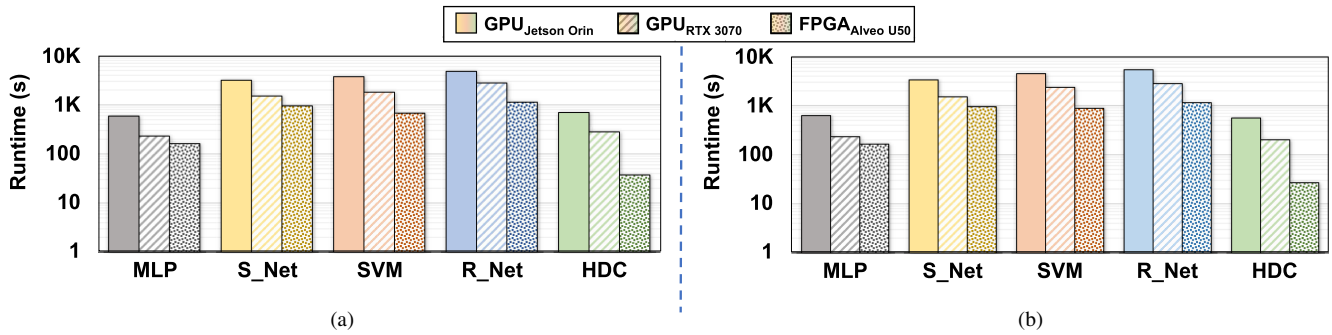


Fig. 12. Training runtime of learning on non-masked IC neural activities for: (a) Consonant classification runtime and (b) vowel classification runtime, where the unrelated information (e.g., vowel part during consonant classification) are not masked.

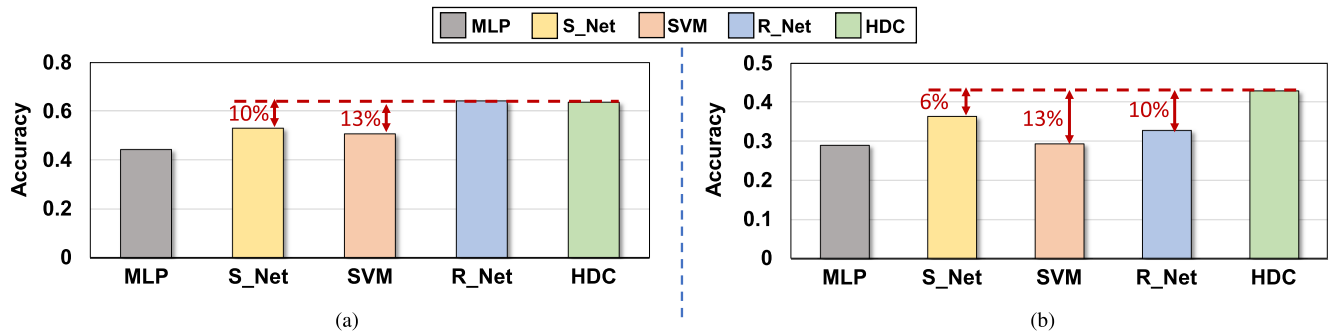


Fig. 13. Accuracy of learning on non-masked IC neural activities for: (a) Consonant classification accuracy and (b) vowel classification accuracy. Our proposed HDC-based algorithm notably improves the accuracy on vowel classification and is comparable to the best baseline in consonant classification.

60 dB SPL clean dataset. As shown in Figs. 12(a) and 13(a), our HDC-based algorithm achieves comparable accuracy compared to ResNet with far less training cost, which makes it significantly outperform other baseline algorithms in consonant classification. In Figs. 12(b) and 13(b), HDC again shows advantages in both accuracy and efficiency. It should be highlighted that our method, when facing non-masked neural activities, achieves notable improvements in vowel classification compared to other baselines, e.g., 10%, 13%, and 6% compared to ResNet, SVM, and SqueezeNet, respectively.

VI. RELATED WORKS

Phoneme Classification on Neural Signals: Mesgarani et al. [15] conducted one of the earliest studies on classifying phonemes based on neural activities. In this study, ECoG recording of ferrets' auditory cortex was used for the phoneme classification task. The linear SVM was chosen as the classifier which achieved up to 37% accuracy for a 14-class consonant classification task. Wang et al. [14] used EEG recordings from four human subjects to perform both consonant and vowel classification tasks. SVM based on multiple sets of manually selected features was used to perform the classification task. The best test accuracy for the 8-class consonant classification task was 66.7% and the best test accuracy for the 4-class vowel classification task was 48.7%. In a recent study [16] aiming to classify the whole set of phonemes in the English language using EEG recordings, the authors achieved the best test accuracy

of 36.1% for 24-class consonant classification. A study done by Armstrong et al. [18] used MUA from gerbils to perform a 12-class consonant classification task, which obtained the best accuracy of 53%. However, these prior works are relatively limited either in task scale, baseline algorithms, or learning accuracy. In this paper, we propose an HDC-based learning framework that supports large-scale neural activity datasets, achieves high learning quality, and is compared to much stronger baselines.

Hyperdimensional Computing: Prior works have applied HDC brain-like memorization and association functionalities to diverse applications, for example, outlier detection [51], speech recognition [10], and gesture classification [12]. Apart from classification learning tasks, it has also been applied to genomic matching [52], regression [53], reinforcement learning [54], and graph reasoning [55], [56]. Particularly, several recent works also propose to leverage HDC for bio-signals (e.g., EEG and EMG signals) processing, including brain-computer interfaces, emotion detection, activity recognition, and gesture detection [12], [30], [57]. Previous HDC works provide comparable accuracy to existing machine learning models while reducing the learning runtime cost. However, they face various challenges when learning on noisy and high-dimensional neural activities.

Hardware Acceleration of Hyperdimensional Computing: Work [41] focuses on accelerating HDC-based regression algorithm on FPGA and work [42] is mainly about accelerating HDC-based Reinforcement Learning (RL) algorithm and

work [10] focuses on HDC classification model for much simpler datasets, including handwritten digits recognition and human activity recognition (based on preprocessed and extracted features). In addition, work [10] focuses on HDC model quantization and puts little emphasis on FPGA on-chip computing IP design. Our design is not only very different from previous designs in terms of targeted applications but also unique architecture-wise.

Architecture-wise, work [10] uses a naive design to parallelize the dot-product computation. It proposes to do element-wise multiplication across all channels and applies an adder tree to accumulate partial results. However, this particular design does not scale well and fails to support biosignals with a large number of channels. Work [10] and work [42] targets RL and regression, where inputs are very small vectors, e.g., inputs are vectors of 4 features for the CartPole RL task. Therefore they select a single large systolic array for computation. In addition, in both RL and regression, the output logit is only a single value, and thus the parallelization there is very straightforward and not optimal for our application.

VII. CONCLUSION

This paper proposes a hyperdimensional learning framework for large-scale neural activity processing. For the first time, we utilize a brain-inspired algorithm for learning on IC neural recordings and perform phoneme recognition with high learning quality and efficiency. Prior HDC-based algorithms only handled neural activities with limited temporal and spatial scales. Learning on large-scale neural activities is non-trivial for these prior methods. On the other hand, our proposed method includes a spatial and temporal-aware HDC encoder that effectively captures global and local patterns without aggressive preprocessing. We carry out extensive experiments across different hardware platforms and show that our design achieves up to $210\times$ speedup against the baseline. In addition, our HDC-based algorithm shows notable benefits over baselines in terms of vowel classification, learning on non-masked neural activity, and learning with the speaker-independent setting.

REFERENCES

- [1] B. P. Bean, "The action potential in mammalian central neurons," *Nature Rev. Neurosci.*, vol. 8, no. 6, pp. 451–465, 2007.
- [2] E. N. Brown et al., "Multiple neural spike train data analysis: State-of-the-art and future challenges," *Nature Neurosci.*, vol. 7, no. 5, pp. 456–461, 2004.
- [3] X. W. Wang et al., "Emotional state classification from EEG data using machine learning approach," *Neurocomputing*, vol. 129, pp. 94–106, 2014.
- [4] Z. Chen et al., "A new algorithm for classification of ictal and pre-ictal epilepsy ECoG using MI and SVM," in *Proc. Int. Conf. Signals Syst.*, 2017, pp. 212–216.
- [5] Y. Zhang et al., "EEG recognition of motor imagery based on SVM ensemble," in *Proc. 5th Int. Conf. Syst. Informat.*, 2018, pp. 866–870.
- [6] Y. R. Tabar and U. Halici, "A novel deep learning approach for classification of EEG motor imagery signals," *J. Neural Eng.*, vol. 14, no. 1, 2017, Art. no. 016003.
- [7] A. Craik et al., "Deep learning for electroencephalogram (EEG) classification tasks: A review," *J. Neural Eng.*, vol. 16, no. 3, 2019, Art. no. 031001.
- [8] L. Alzubaidi et al., "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, pp. 1–74, 2021.
- [9] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cogn. Computation*, vol. 1, no. 2, pp. 139–159, 2009.
- [10] M. Imani et al., "Revisiting hyperdimensional learning for FPGA and low-power architectures," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2021, pp. 221–234.
- [11] A. Rahimi et al., "Hyperdimensional computing for blind and one-shot classification of EEG error-related potentials," *Mobile Netw. Appl.*, vol. 25, pp. 1958–1969, 2020.
- [12] A. Rahimi et al., "Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition," in *Proc. IEEE 2016 Int. Conf. Rebooting Comput.*, 2016, pp. 1–8.
- [13] A. Burrello et al., "One-shot learning for iEEG seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing," in *Proc. IEEE 2018 Biomed. Circuits Syst. Conf. (BioCAS)*, 2018, pp. 1–4.
- [14] R. Wang et al., "Using phase to recognize English phonemes and their distinctive features in the brain," *Proc. Nat. Acad. Sci. United States Amer.*, vol. 109, no. 50, pp. 20685–20690, 2012.
- [15] N. Mesgarani et al., "Phoneme representation and classification in primary auditory cortex," *J. Acoust. Soc. Amer.*, vol. 123, no. 2, pp. 899–909, 2008.
- [16] E. M. Mugler et al., "Direct classification of all American English phonemes using signals from functional speech motor cortex," *J. Neural Eng.*, vol. 11, no. 3, 2014, Art. no. 035015.
- [17] M. S. Lewicki, "A review of methods for spike sorting: The detection and classification of neural action potentials," *Netw.: Computation Neural Syst.*, vol. 9, no. 4, 1998, Art. no. R53.
- [18] A. G. Armstrong et al., "Compression and amplification algorithms in hearing aids impair the selectivity of neural responses to speech," *Nature Biomed. Eng.*, vol. 6, pp. 717–730, 2021.
- [19] H. Spoendlin and A. Schrott, "Analysis of the human auditory nerve," *Hear. Res.*, vol. 43, no. 1, pp. 25–38, 1989.
- [20] P. Heil and A. J. Peterson, "Basic response properties of auditory nerve fibers: A review," *Cell Tissue Res.*, vol. 361, pp. 129–158, 2015.
- [21] J. K. Moore and K. K. Osen, "The cochlear nuclei in man," *Amer. J. Anatomy*, vol. 154, no. 3, pp. 393–417, 1979.
- [22] L. Ouda and J. Syka, "Immunocytochemical profiles of inferior colliculus neurons in the rat and their changes with aging," *Front. Neural Circuits*, vol. 6, pp. 1–14, 2012.
- [23] S. Waldert et al., "A review on directional information in neural signals for brain-machine interfaces," *J. Physiol.-Paris*, vol. 103, no. 3–5, pp. 244–254, 2009.
- [24] A. Huet et al., "Sound coding in the auditory nerve of Gerbils," *Hear. Res.*, vol. 338, pp. 32–39, 2016.
- [25] A. Rees et al., "Regularity of firing of neurons in the inferior colliculus," *J. Neurophysiol.*, vol. 77, no. 6, pp. 2945–2965, 1997.
- [26] C. J. Stoodley, "The cerebellum and cognition: Evidence from functional imaging studies," *Cerebellum*, vol. 11, no. 2, pp. 352–365, 2012.
- [27] K. B. McDermott et al., "A procedure for identifying regions preferentially activated by attention to semantic and phonological relations using functional magnetic resonance imaging," *Neuropsychologia*, vol. 41, no. 3, pp. 293–303, 2003.
- [28] R. Gayler, "Multiplicative binding, representation operators and analogy," in *Advances in Analogy Research: Integration of Theory and Data From the Cognitive, Computational and Neural Sciences*. Sofia, Bulgaria: New Bulgarian University, 1998.
- [29] A. Moin et al., "An EMG gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier," in *Proc. IEEE 2018 Int. Symp. Circuits Syst. (ISCAS)*, 2018, pp. 1–5.
- [30] E.-J. Chang et al., "Hyperdimensional computing-based multimodality emotion recognition with physiological signals," in *Proc. IEEE 2019 Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, 2019, pp. 137–141.
- [31] S. Kiranyaz et al., "Real-time patient-specific ECG classification by 1-D convolutional neural networks," *IEEE Trans. Biomed. Eng.*, vol. 63, no. 3, pp. 664–675, Mar. 2016.
- [32] M. Imani et al., "DUAL: Acceleration of clustering algorithms using digital-based processing in-memory," in *Proc. 2020 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2020, pp. 356–371.
- [33] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, vol. 20, pp. 1177–1184.
- [34] B. Schölkopf, "The kernel trick for distances," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, vol. 13, pp. 283–289.
- [35] C. A. Micchelli et al., "Universal kernels," *J. Mach. Learn. Res.*, vol. 7, no. 12, pp. 2651–2667, 2006.

- [36] S. Salamat et al., "F5-HD: Fast flexible FPGA-based framework for refreshing hyperdimensional computing," in *Proc. 2019 ACM/SIGDA Int. Symp. Field- Program. Gate Arrays*, 2019, pp. 53–62.
- [37] M. Imani et al., "SparseHD: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *Proc. 2019 IEEE 27th Annu. Int. Symp. Field- Program. Custom Comput. Machines (FCCM)*, 2019, pp. 190–198.
- [38] T. Zhang et al., "HD2FPGA: Automated framework for accelerating hyperdimensional computing on FPGAs," in *Proc. 2023 24th Int. Symp. Qual. Electron. Des. (ISQED)*, 2023, pp. 1–9.
- [39] Y. Thoma et al., "FPGA-GPU communicating through PCIe," *Microprocessors Microsystems*, vol. 39, no. 7, pp. 565–575, 2015.
- [40] J. Wang, L. Guo, and J. Cong, "AutoSA: A polyhedral compiler for high-performance systolic arrays on FPGA," in *Proc. 2021 ACM/SIGDA Int. Symp. Field- Program. Gate Arrays*, 2021, pp. 93–104.
- [41] H. Chen et al., "Full stack parallel online hyperdimensional regression on FPGA," in *Proc. 2022 IEEE 40th Int. Conf. Comput. Des. (ICCD)*, 2022, pp. 517–524.
- [42] H. Chen et al., "DARL: Distributed reconfigurable accelerator for hyperdimensional reinforcement learning," in *Proc. 41st IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2022, pp. 1–9.
- [43] P. Poduval et al., "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *Proc. 2021 58th ACM/IEEE Des. Automat. Conf. (DAC)*, 2021, pp. 781–786.
- [44] A. Hernandez-Cane et al., "OnlineHD: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Des., Automat. Test Europe Conf. Exhib. (DATE)*, 2021, pp. 56–61.
- [45] F. N. Iandola et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 mb model size," 2016, *arXiv:1602.07360*.
- [46] K. He et al., "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
- [47] Z. Wen et al., "ThunderSVM: A fast SVM library on GPUs and CPUs," *J. Mach. Learn. Res.*, vol. 19, no. 1, pp. 797–801, 2018.
- [48] A. J. A. El-Maksoud, "FPGA design of high-speed convolutional neural network hardware accelerator," in *2021 3rd Novel Intell. Leading Emerg. Sci. Conf. (NILES)*, 2021, pp. 376–379.
- [49] M. B. Rabieah and C.-S. Bouganis, "FPGASVM: A framework for accelerating kernelized support vector machine," in *Proc. Workshop Big Data, Streams Heterogeneous Source Mining: Algorithms, Syst., Program. Models Appl.*, 2016, pp. 68–84.
- [50] S. Lindemann, "Variation or 'error'? perception of pronunciation variation and implications for assessment," in *Second language Pronunciation Assessment*. Bristol, U.K.: Multilingual Matters, 2017, pp. 193–209.
- [51] R. Wang et al., "ODHD: One-class brain-inspired hyperdimensional computing for outlier detection," in *Proc. 59th ACM/IEEE Des. Automat. Conf.*, 2022, pp. 43–48.
- [52] Y. Kim et al., "GenieHD: Efficient dna pattern matching accelerator using hyperdimensional computing," in *Proc. IEEE 2020 Des., Automat. Test Europe Conf. Exhib. (DATE)*, 2020, pp. 115–120.
- [53] A. Hernández-Cano et al., "Hyperdimensional computing with holographic and adaptive encoder," *Front. Artif. Intell.*, vol. 7, 2024, Art. no. 1371988.
- [54] Y. Ni et al., "HDPG: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proc. 59th ACM/IEEE Des. Automat. Conf.*, 2022, pp. 1141–1146.
- [55] I. Nunes et al., "GraphHD: Efficient graph classification using hyperdimensional computing," in *Proc. 2022 Des., Autom. Test Europe Conf. Exhib. (DATE)*, 2022, pp. 1485–1490.
- [56] P. Poduval et al., "GraphHD: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Front. Neurosci.*, vol. 16, 2022, Art. no. 757125.
- [57] U. Pale et al., "Systematic assessment of hyperdimensional computing for epileptic seizure detection," in *Proc. IEEE 2021 43rd Annu. Int. Conf. Eng. Med. Biol. Soc. (EMBC)*, 2021, pp. 6361–6367.