HyperSpikeASIC: Accelerating Event-based Workloads with HyperDimensional Computing and Spiking Neural Networks

Tianqi Zhang, Justing Morris, Kenneth Stewart, *Graduate Student Member, IEEE*, Hin Wai Lui, Behnam Khaleghi, Anthony Thomas, Thiago Goncalves-Marback, Baris Aksanli, *Member, IEEE*, Emre O. Neftci, *Member, IEEE*, and Tajana Rosing, *Fellow, IEEE*

***Abstract*—Today's Machine Learning(ML) systems, running workloads such as Deep Neural Networks, which require billions of parameters and many hours to train a model, consume a significant amount of energy. Due to the complexity of computation and topology, even the quantized models are hard to deploy on edge devices under energy constraints. To combat this, researchers have been focusing on new emerging neuromorphic computing models. Two of those models are Hyperdimensional Computing (HDC) and Spiking Neural Networks (SNNs), both with their own benefits. HDC has various desirable properties that other Machine Learning (ML) algorithms lack such as robustness to noise, simple operations, and high parallelism. SNNs are able to process event-based signal data in an efficient manner. This work develops HyperSpike, which utilizes a single, randomly initialized, and untrained SNN layer as a feature extractor connected to a trained HDC classifier. HDC is used to enable more efficient classification as well as provide robustness to errors. We experimentally show that HyperSpike is on average $31.5\times$ more robust to errors than traditional SNNs. On Intel's Loihi [1], HyperSpike is $10\times$ faster and $2.6\times$ more energy efficient over traditional SNN networks. We further develop HyperSpikeASIC, a customized accelerator for HyperSpike. By decoupling the neuron and synapses, HyperSpikeASIC skips the inactive neurons and limits the neuron state updating to once per time step at most. HyperSpikeASIC is $601\times$ faster and $3467\times$ more energy efficient than HyperSpike running on Intel's Loihi for SNN acceleration, and $12.2\times$ faster and $211\times$ more energy efficient than the state-of-the-art SNN ASIC implementation [2].**

I. Introduction

The Internet of Things (IoT) era put more stringent demands on battery lifetime due to a large number of compute-intensive machine learning algorithms that are deployed on edge devices. The slowdown of Moore's Law has made this challenge even greater. Some designers are pinning their hopes on using cloud computing to centralize state-of-the-art machine learning models and distribute the results to edge devices. This raises concerns for privacy-conscious users, adds a lot of communication overhead, and prevents the deployment of real-time tasks.

Neuromorphic computing models and corresponding accelerators are currently being explored as an alternative to classical machine learning for edge devices. Their key strength is unprecedented energy efficiency, which makes them an excellent match for power-limited applications such as those used in the IoT. Two such promising neuromorphic computational models are Hyperdimensional Computing (HDC) and Spiking Neural Networks (SNNs).

SNNs are neuromorphic models that mimic the dynamics of biological neurons via spike-based communication mechanism [3]. Such spike-based information transfer can implicitly utilize the inter-spike time interval to efficiently carry information, decreasing the signal processing energy consumption [4]. The SNN can be trained with backpropagation-based or plasticity-based algorithms. Due to the activation of spiking neurons being binary and thus non-differentiable, in addition to the processing component of SNNs being temporal, it's difficult to train SNN models directly with backpropagation-based methods [5]. Some approaches use a surrogate gradient with smoothed activation function for error gradients [6], [7]. Plasticity-based algorithms, like Spike-timing-dependent plasticity (STDP), adjust the weights depending on the interval of relative spike timings from presynaptic and postsynaptic neurons [8], [9]. Both require a nonlinear function and at least one addition tracing variable for each synaptic to update the weights, whose number is proportional to the square of the number of neurons [7]. Some emerging neuromorphic sensors, such as Dynamic Vision Sensor (DVS), can sample the signal and encode it into temporal signal directly for SNN processing [10]. However, the lower signal-to-noise ratio of these novel sensors, coupled with the randomness of spiking coding and the storage limitation of SNN network weights, pose challenges for real-world deployment [11], [12].

Brain-inspired Hyperdimensional Computing (HDC) uses high dimensional vectors, *hypervectors* (HV), to represent signals. Unlike traditional processing systems where information is quantized and encoded into single digital values, HDC encodes data into hypervectors, and then combines them to carry information [13]. HDC uses simple, easy-to-parallelize operations for learning (addition, multiplication, permutation, nearest neighbor search) [14], and as such, is a perfect match for hardware acceleration [15], [16]. HDC has been successfully applied to many classification tasks, such as activity recognition, face detection, language recognition,

This work was supported in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA, in part by SRC-Global Research Collaboration grant Task No. 2988.001, and also NSF grants 1527034, 1730158, 1826967, 1830331, 1911095, 2003277, 1652159, 2003279, 2100237, 2112167, 2052809, 2112665.

T. Zhang, B. Khaleghi, A. Thomas, T. Marback, and T. Rosing are with the University of California, San Diego (UCSD), La Jolla, CA 92093 USA (e-mail: tiz014@ucsd.edu; bkhalegh@eng.ucsd.edu; ahthomas@eng.ucsd.edu; tmarback@ucsd.edu; tajana@eng.ucsd.edu).

J. Morris is with California State University San Marcos (CSUSM), San Marcos, CA 92096 USA (e-mail: hwlui@uci.edu; eneftci@uci.edu).

K.Stewart, H. Lui and E. Neftci are with the University of California, Irvine (UCI), Irvine, CA 92697 USA (e-mail: kennetms@uci.edu, hwlui@uci.edu; eneftci@uci.edu).

B.Aksanli is with the San Diego State University (SDSU), San Diego, CA 92182 USA (e-mail: baksanli@sdsu.edu).

image classification [17], [18], [15], as well as clustering[19], [20], recommendation systems[21], and others[22], [23]. HDC can be trained in one shot with little to no retraining needed. It stores only the trained class HVs, thus dramatically reducing the storage needs compared to neural networks, for example, [24] shows only 3.66% storage is required compared to the neural network with the same accuracy for the speech recognition task. HDC is also more robust to noise because of its high dimensionality [25]. However, for some types of data, such as large images and event-based data, HDC needs an additional feature extractor before encoding the data into hypervectors to achieve state-of-the-art accuracy.

HyperSpike combines *a single untrained* SNN convolution layer with HDC: the randomized SNN layer extracts the features which are then encoded into hypervectors for training by the HDC layer. This completely removes the need for expensive SNN training [5], while significantly enhancing robustness via HDC classification layer. Our recent work showed that HyperSpike using Intel's Loihi platform and showed comparable accuracy to the state-of-the-art SNNs, with improved performance by $10\times$, reduced energy consumption by $2.6\times$, while being $31.5\times$ more robust to errors than SNNs [26].

In this work, we develop an accelerator for HyperSpike, called HyperSpikeASIC, and test it with a wider variety of workloads and network structures: HyperSpike could be applied with not only a convolution layer for event-based image signals but also with a fully-connected layer for 1D signals with sigma-delta modulation, thus making it possible to run both DVS camera and EMG workloads. Our design takes event-based coding and decouples synapse state update and neuron state update using efficient independent process elements (PEs) to skip inactive neurons update: synapses PEs update the synapse states first and accumulate them, while the neuron PEs only update the active neurons. We trade off the ratio of the two between throughput and area. Compared with HyperSpike using Intel's Loihi for SNN acceleration, HyperSpikeASIC implements SNN and HDC on the same chip to eliminate inter-chip communication and achieves $3,657\times$ higher throughput area ratio (or $601\times$ on delay) and $3,467\times$ more energy efficiency. Compared with the state-of-the-art SNN accelerator[2], we achieve $98\times$ boost on throughput area ratio (or $12.2\times$ on delay) and $211\times$ more energy efficiency.

II. Background and Related Work

A. Spiking Neural Networks

SNNs can be formulated as a type of recurrent neural network with binary activation functions [27]. With this formulation, SNN training can be carried out using standard tools of auto differentiation. To best match the dynamics of existing digital neuromorphic hardware implementing SNNs [28], our neuron model consists of a discretized Leaky Integrate and Fire (LIF) neuron model with a time step Δt written in the form of a Spike Response Model [29]:

$$\begin{aligned} P_j^{t+\Delta t} &= \alpha P_j^t + S_{in}^t, \\ R_i^{t+\Delta t} &= \alpha R_i^t + \alpha U_i^t S_i^t, \\ U_i^{t+\Delta t} &= \sum_j W_{ij} P_j^{t+\Delta t} - R_i^{t+\Delta t}, \\ S_i^{t+\Delta t} &= \Theta(U_i^{t+\Delta t}). \end{aligned} \tag{1}$$

where the constant $\alpha = \exp(-\frac{\Delta t}{\tau_{\text{mem}}})$ reflects the decay dynamics of the membrane potential during a Δt timestep, where τ_{mem} and is the membrane time constant. The time step in our experiments was fixed to $\Delta t = 1$ms, which has been demonstrated to achieve the balance between computational complexity and accuracy [7]. R_i here represents the reset and refractory period of the neuron, and state P_i is the pre-synaptic trace that captures the leaky dynamics of the membrane potential. $S_i^t = \Theta(U_i^t)$ represents the spiking non-linearity, computed using the unit step function, where $\Theta(U_i) = 0$ if U_i is smaller than threshold U_{th}, otherwise 1. We distinguish here the input spike train S_{in}^t from the output spike train S^t. For the purposes of computing the gradient, the derivative of Θ is replaced with the derivative of a smooth function, the fast sigmoid function [30], following the surrogate gradient approach [27].

SNN accelerators use Pseudo Random Number Generators (PRNGs) when choosing the routing path [31], adding noise to the fire threshold [32][4], modifying the weight changing probability [33], and converting the coding format [34]. Very few works take advantage of this randomness for feature extraction. Single or multiple fully connected layer models are widely used [4][33][34][35] in SNN accelerators. However, a fully connected layer requires storing more weight than a convolutional layer. The weights are stored in the scratchpad memory or on each PE locally, which takes the majority of the area [31]. Binary-weight SNN accelerators [33][34][36] have been proposed to reduce the size of the weight memory but require more layers for equivalent accuracy. Spike convolutional neural network (SCNN) models were adopted in [37] and [38] and implemented on the TrueNorth [39] chip. Following the orderliness of CNN dataflow, some SCNN accelerators scan the pixels of SCNN inputs and do not consider the fact that only some neurons fire at a time and thus do not get the benefits from the spatial sparsity of inputs [40]. The SNN accelerators are also customized for the models that only use SNN as a feature extractor, like in SAILnet spares coding algorithm [41] and Liquid State Machine (LSM) [2]. In HyperSpikeASIC, we use HDC as a classifier to reduce the storage requirement of SNN weights and the time and complexity needed for training. We use a randomly initialized layer of the SNN as a feature extractor to convert the DVS signal to HDC-friendly feature vectors. The proposed architecture takes advantage of spike sparsity for event-driven input to skip inactive neuron state updates. It avoids the need for training the SNN by using HDC's lightweight training instead. Through HDC it also gains the benefit of high noise robustness.

B. Hyperdimensional Computing

HDC models the sparse distributed memory, which is a mathematical representation of human memory, with spare high-dimensional vectors, by mimicking the brain's neural network[42]. There are three major steps for HDC: encoding, training, and inference. Encoding maps input data into high-dimensional vectors with associative algebra. In HD space, each dimension is expected to carry independent information. The high redundancy of HVs makes it less sensitive to noise and thus being widely explored for scenarios requiring robustness [43]. The training stage combines HVs from the

same class and creates a model storing the class HVs. During inference, HDC checks the similarity between the incoming sample's HV and the trained model to find the most similar class.

(1) Encoding: Let us assume a feature vector $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, with n features in the original domain. The goal of encoding is to map this feature vector to a D dimensional space vector: $\mathcal{H} = \{h_1, h_2, \ldots, h_D\}$. The encoding first randomly generates D dense bipolar vectors with the same dimensionality as the original domain, $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_D\}$, where $\mathbf{p}_i \in \{-1, 1\}^n$. Many different projection matrices have been purposed as different encoding methods, for example, BRIC only keeps several elements near the diagonal to reduce encoding computational intensity on large-scale HV [18]. Nonlinear operators, like shuffle, are also introduced into encoding for times series data [44]. As there is no additional feature extractor in those works, the fine-designed encoding stage is critical to the accuracy. In this work, our HDC only works as a classifier and thus we use the simple randomly generates bipolar vectors as projection vectors. The inner product of a feature vector with each generated vector gives us a single dimension of a hypervector (HV) in high-dimensional space. To encode a feature vector into an HV, we perform a matrix-vector multiplication between the projection matrix and the feature vector using:

$$\mathcal{H} = sign(\mathcal{P} \times \mathcal{F}) \tag{2}$$

where $sign(\cdot)$ is a sign function that maps the result of the dot product to +1 or -1.

(2) Training: HD computing is very easy to train - only addition is needed to create class hypervectors. After encoding, the original samples are represented as hypervectors. Consider HV $\mathcal{H}_i$ as the encoded HV of input i with the procedure explained above. Let the label of input i is l_i. HD training simply adds all hypervectors belong to the same class to generate the final class hypervector. Specifically, the class HV of label j, denoted by $\mathcal{C}_j$, is:

$$\mathcal{C}_j = \sum_{i \in \{i | l_i = j\}} \mathcal{H}_i. \tag{3}$$

In contrast to the simple training that HDC requires, neural networks require complex backpropagation algorithms to train.

(3) Similarity checking: The inference searches for the most similar class HV to the encoded query. When hypervectors are binary, such as in this work, the search is done using Hamming distance. When HVs are not binary, cosine similarity is used instead. Assume the pairwise vector distance metrics $Dist(a, b)$ is used. The predicted label of query vector $\mathcal{H}$, denoted by $\hat{l}$ is

$$\hat{l} = \underset{j}{\operatorname{argmin}}\ Dist(\mathcal{H}, \mathcal{C}_j) \tag{4}$$

HDC is lightweight enough to run with acceptable performance on CPUs [45]. However, utilizing a parallel architecture can significantly speed up HDC execution time. Imani *et al.* showed two orders of magnitude speed up when HD runs on GPU [46]. Salamat *et al.* proposed a framework that facilitates fast implementation of HDC algorithms on an FPGA [47]. HDC ASIC accelerators provide significant improvements in performance [14] due to the bit-level operations that are easily accelerated in silicon. There have been multiple works on implementing HDC on new emerging computing hardware such as ReRAM crossbar [48]. However, these works assume an older model of ReRAM and do not consider various sources of errors common to ReRAM. In this paper, we accelerate HyperSpike in an ASIC (see Section IV) and show that our design is very robust to significant levels (3.4%) of bit error rates (see Section V-B).

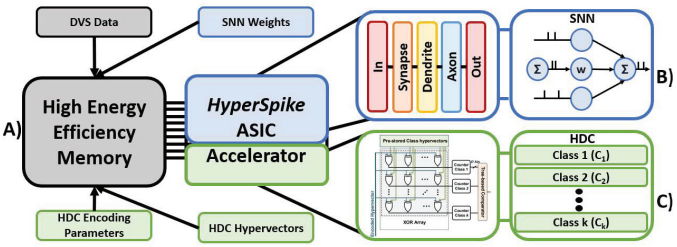


Fig. 1. Overall System Architecture of HyperSpike.

Several works show that HDC is inherently robust to noise [49], [50], [25]. Work in [49] investigated the robustness of HDC to RTL errors and found that the HDC-based approach tolerated $8.8\times$ higher probability of bit-level errors, similar to [50]. Work in [25] showed that HDC is also robust to wireless communication errors.

The semantic pointer architecture implements a type of HD-computing using SNNs and the Neural Engineering Framework [51] . Since then other approaches to using semantic pointer architectures for SNN computation have been proposed such as representing phasors as spike times [52] and representing hypervectors as Sparse Block-Codes [53]. However, none of these works have been implemented in hardware[54].

HyperSpike differs from previous approaches as it links two separate blocks, namely SNNs and HD computing, rather than implementing SNNs using HD or vice versa. Our initial implementation for HyperSpike [26] used the Intel Loihi [28] to evaluate the neuromorphic processing component. The Intel Loihi is a 60-mm^2 neuromorphic processor fabricated in Intel's 14-nm process that integrates a wide range of features such as hierarchical connectivity, dendritic compartments, synaptic delays, and programmable synaptic learning rules [28]. Results from previous studies have demonstrated that brain-inspired networks implemented on the Intel Loihi, such as SNNs, using precise spike-timing relationships from event-based data processing, perform certain computations with orders of magnitude lower latency and energy compared to the conventional state-of-the-art approaches such as those based on feedforward deep neural networks [55]. Loihi is also capable of low-power edge learning without needing a cloud [3]. It uses SRAM for the state, causing the state bit cost to be higher than conventional processors. To reduce the cost of the Loihi chip one could store state in ReRAM but at the cost of higher bit error rates [56].

III. Motivations and Benefits of SNNs with HDC

This section shows that neither SNN nor HDC model can perform well when analyzing event-based tasks in high error regimes. Therefore, we combine SNN with HDC to create HyperSpike. In addition to overcoming those challenges,

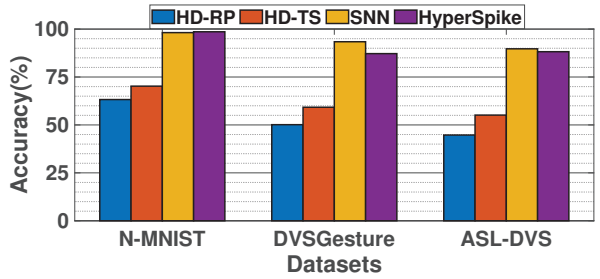
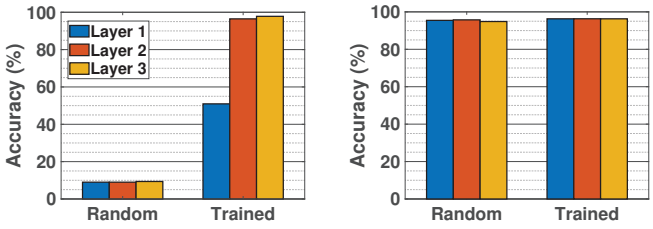


Fig. 2. Comparison of Using Existing HDC vs Traditional SNNs



(a) SNN-MLP (b) HyperSpike

Fig. 3. Impact of Random Weights on HyperSpike and Traditional SNNs Across Different Layers of the SNN using the N-MNIST dataset.

HyperSpike only requires one SNN layer and doesn't need expensive SNN training. Figure 1 provides a high-level overview of HyperSpike. High energy efficiency memory (A) can be used by HyperSpike to store the parameters of the SNN and HDC layers. The high energy efficiency memory, such as ReRAM and MRAM, is typically at the cost of reliability [56]. We will demonstrate that the HyperSpike is less sensitive to memory errors in Section V-B. But in the ASIC design, we still used SRAM generated by the memory compiler. The SNN algorithm is shown in part (B), while HDC is in part (C). HyperSpikeASIC accelerates both SNN and HDC in an ASIC. The details of our hardware accelerator are provided in Section IV.

HyperSpike combines SNN as a feature extractor and HDC as a classifier. The SNN processes the event-based spikes at the input and generates a feature vector at the last time step of each read. Our results in Figure 3 show that HyperSpike only requires a single layer untrained SNN. The HDC encodes the feature vector from the output of the SNN and is trained. We used the online training method discussed in SectionII-B for class HVs. Then, for inference, the encoded HV is compared to the class HVs using the Hamming distance metric.

HDC cannot process event-based data alone: Figure 2 shows the accuracy of HDC on three event-based datasets which are collected by DVS directly [57], [58], [59]. We tested two encoding methods: one that targets feature vectors using random projection (HD-RP) [18] and the other using the encoding method for times series data (HD-TS) [44]. With an average HDC accuracy of 57%, compared to an average accuracy of 94% on SNNs, we can see that existing HDC systems that do not use SNN-based feature extraction cannot classify DVS data as accurately. To combat this, we need a feature extractor to represent the event-based data better. The following experiments show that SNN can transform the event-based data into feature vectors that HDC can encode and use for accurate classification.

The HD-RP generates the projection matrix $\mathcal{P}$ randomly. Each element of the $\mathcal{P}$ follows the Bernoulli distribution. The HD-TS will consider the timestamp of feature f_i. Assume the timestamp of feature f_i is t_{f_i}. The jth column of projection matrix $\mathcal{P}$, denoted by $\overrightarrow{P_{.,j}}$ is

$$\overrightarrow{P_{.,j}} = \rho^{t_{f_j}} \overrightarrow{g_{.,j}} \tag{5}$$

where g_{ij} is random constant coefficient and ρ is permutation operation. ρ^k means to rotate a vector k times. For example, $[a,b,c,d]^T\rho = [b,c,d,a]^T, [a,b,c,d]^T\rho^2 = [c,d,a,b]^T$. Notice for different sample, the projection matrix $\mathbf{P}$ for HD-RP is same but for HD-TS is different because the timestamp t_{f_i} of feature f_i may vary among different sample.

SNN requires multiple layers for acceptable accuracy alone Figure 3a shows the accuracy of SNN-MLP with different numbers of layers on the N-MNIST dataset. The SNN is trained with the backpropagation-based algorithm DECOLLE [7]. We can see that at least 2 layers are required to get acceptable accuracy for SNN models. The SNN results in figure 2 adopting 3-layer trained SNN. The configuration of SNN models for different datasets is shown in TableII.

SNN is sensitive to in memory errors: When a query comes in for processing on HyperSpike, the data is first stored in the memory shown as (A) in Figure 1. Previous designs with emerging computing hardware assumed that the technology had ideal characteristics and no errors, which is not true in general. We evaluate bit-level errors in Section V. These errors extend to all data stored in memory, such as SNN weights and hypervectors for HDC. However, as we describe below, HDC is able to overcome these errors due to its robustness. Although we model our memory separately from our compute chip, our experiments can be generalized to other applications where bit error rates occur on the model parameters. For instance, our results would extend to HyperSpike storing model parameters and computing in ReRAM. This can be further generalized to other hardware with bit error rates (BER), such as emerging non-volatile memories, low voltage memories, or even wireless communication [56].

HyperSpike Removes the Need for SNN Training: Figure 3 compares the accuracy results of using random weights and trained weights with both HDC as the classifier and a more traditional MLP as the classifier on N-MNIST. The results show that when using a traditional classifier, such as an MLP, training of the SNN weights is necessary. In contrast, HDC is able to achieve high accuracy even with random SNN weights. We got similar results for all tested datasets.

HyperSpike Only Needs One Untrained SNN Layer: Figure 3 also shows the accuracy of attaching HDC to different layers of the SNN on N-MNIST. The data in Figure 3b shows that HDC attached to SNN layers is able to achieve high accuracy across all layers of the attached SNN, unlike a standalone trained SNN shown in Figure 3a that only achieves high accuracy in the last layer. In fact, we can see that HDC achieves comparable accuracy to its maximum accuracy with just one SNN layer. This trend is true across all the datasets we tested. Therefore, HyperSpike utilizes just one SNN layer as a preprocessing step to transform the data from DVS data to a feature vector. This allows HyperSpike to save even more energy over traditional SNN networks.

IV. HyperSpikeASIC Design

This section introduces HyperSpikeASIC, which accelerates

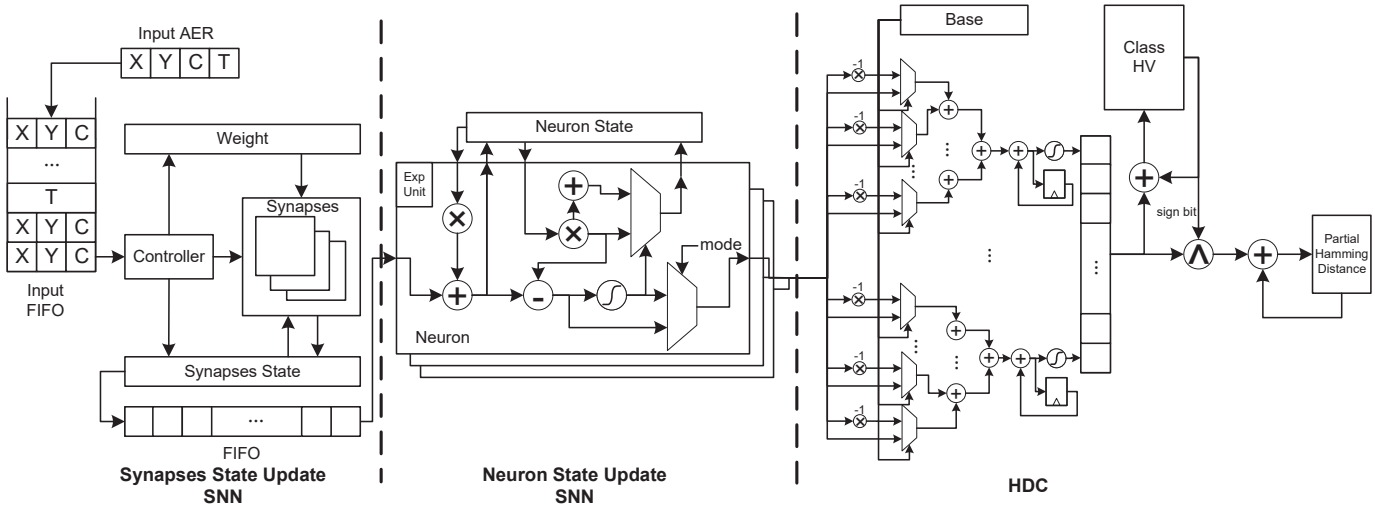


Fig. 4. Hardware Architecture

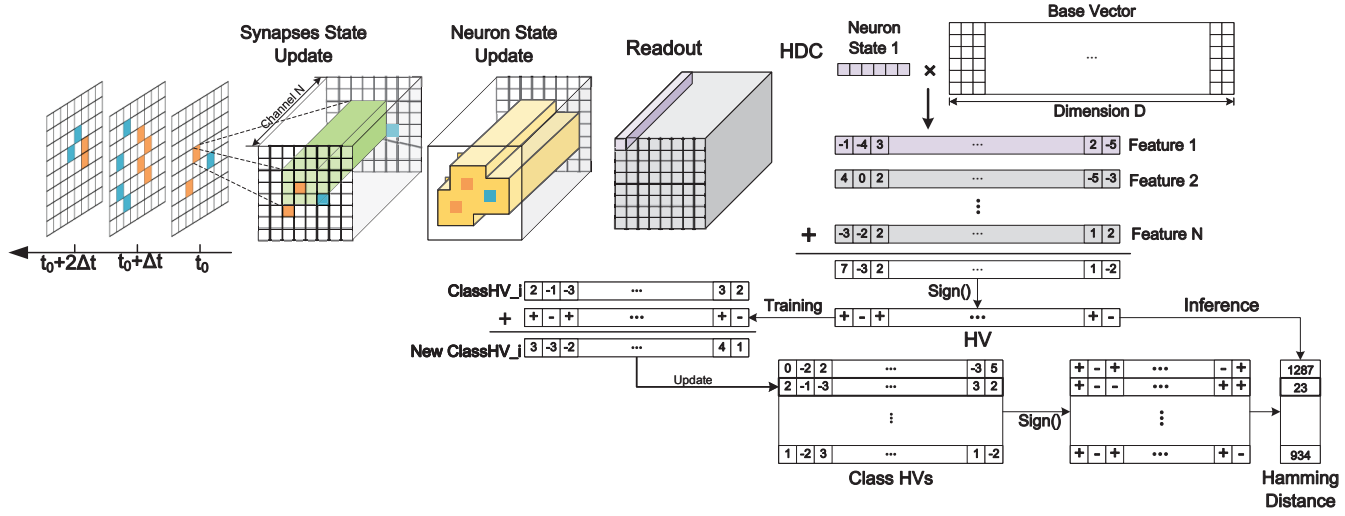


Fig. 5. Example HyperSpikeASIC Dataflow

HyperSpike. Figure 4 shows the proposed accelerator architecture. The first two stages, synapse state update, and neuron state update are a part of the SNN hardware and the last stage is HDC. The SNN accelerator has (i) decoupled synapse process elements (PEs) and neuron PEs to support a highly efficient convolutional layer and (ii) variable precision neuron state update to increase throughput and provide enough precision for HDC. The HDC accelerator uses INT8 hypervectors for training and binary hypervectors with Hamming distance for inference. HyperSpikeASIC fetches data from memory and performs the SNN layer operations first. It supports SNN models with up to 32K neurons in order to support a variety of applications. The computations use 16-bit fixed-point precision [60] to reduce the impact of the bit errors.

As shown in Figure 4, HyperSpikeASIC has three stages: synapses and neuron state update, and HDC. The first two stages are SNN processing for feature extraction, while the last one is HDC for classification. The input data are fetched from the input FIFO and decoded. When a pre-synapse neuron fires, the states of synapses need to be updated. The synapses PEs accumulate the corresponding weights and store the intermediate result in the synapse state memory. The detail about synapses state update is discussed in Section IV-A. When the controller detects that an input belongs to a new time step, the updated synapse states are written into the buffer along with the addresses of the neuron connected to the fired input neuron. The neuron PEs use that information from the buffer to update the neuron state following the procedure talked about in Section IV-B. The number of synapse PEs, N_{SYN}, and neuron PEs, N_{NEU}, is different because there are typically more synapse states required to update than neuron states. When one frame is done, the neuron PEs fetch all the neuron states and work in readout mode (see Section IV-B) to generate the features for HDC. HDC maps those states to HVs and compares them with the stored class hypervectors to find the closest class (see Section IV-D).

Figure 5 shows an example of processing stages. The most front of HyperSpikeASIC is the synapse state update module, which receives the event-based inputs and processes them spike by spike. The spikes that fire in the same time step aggregate to the synapse state. Only partial post-synaptic neurons would receive the updated synapse states. The neuron state module updates time step by time step and skips the inactive neurons. In detail, the event that occurs at time t

always comes before the event occurs at time t+d. However, if two events occur in the time range $[t, t+w)$ and thus are treated as occurring in the same time step, the event at the bottom of the sensor may come before or after the event at the top of the sensor. The update frequency is further reduced for HDC as HDC only readouts the neuron state once per sample. The first stage of HDC is to map the neuron state to HVs no matter training or inference. And then the HDC training would accumulate the encoded HV to the corresponding class HV while HDC inference searches for the most similar class HV to the encoded query. There are 3 input spikes at time t_0, two of which are positive events (shown as orange) while one is a negative event (shown as blue). As the input is event-based, we may receive the spikes chronologically but out of order on spatial. Let's assume we have received and processed the negative and left-down positive events and will process the right-up positive event. The figure 5 shows that each pre-synaptic neuron connects to $9N$ post-synaptic neuron, where N is the channel size. That is, the synapse state located in the green region will be updated. As it is the last spike fired at t_0, the next step is to update the neuron states. Each neuron state would be updated once at most. In this example, the neuron in the yellow region, the union set of neurons that received spikes, will be updated. After finishing processing the last spike belonging to this input sample, it would readout each neuron's state to generate the HVs. Limited by the bandwidth of neuron state storage, we fetch them along channel dimension, shown as the purple vector. The HDC encoding is matrix multiplication. The base vectors matrix is bipolar, which means only add/minus operators are necessary. HVs generated by different features (neuron states) of the same sample are bonded by accumulation. And then, we use the sign of it as the final HV. The HD training is supervised: fetch the class HV corresponding to the label and update it following equation (3). The HD inference will search for the most similar class HV under Hamming distance. Noted that we use binary HDC; we will only use the sign bits of class HVs.

Figure 6 shows two types of pipeline scheduling of SNN: with or without combining spikes. We process the data frame by frame in the first approach and spike by spike in the second approach. Due to the post-synaptic neuron usually receiving more than one spike at each time step, the neuron PEs may be idle for a long time if there are as many neuron PEs as synapses PEs. Therefore, by combining the spikes that fire in the same time step's spikes by accumulating them to the intermediate synapse state, we can explore the optimized configurations of the ratio of the number of synapses PEs to the number of neuron PEs, which is discussed in Section V-C.

A. Synapses State Update

SNN encoding: The SNN may take rate coding or temporal coding. For rate coding, the encoded value is represented as the rate of spiking, which needs a higher spiking rate to guarantee enough information is represented compared to temporal coding. Some experimental results find the rate coding in sensory and motor systems [61]. Some works contributed to converting the ANN to SNN take rate coding and proving the conversion error can be 0 [62], [63], [64]. Rate coding is simple

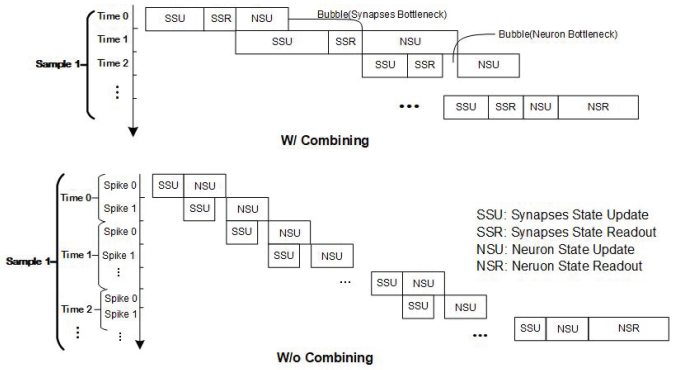


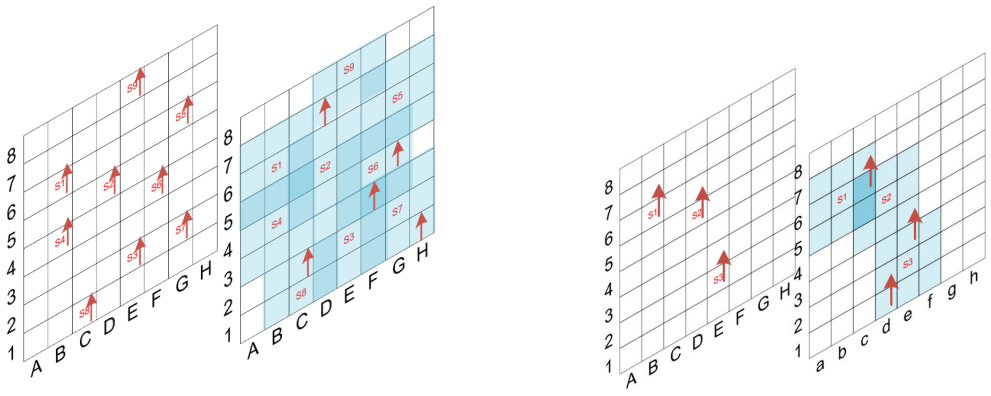
Fig. 6. Pipeline Scheduling of SNN

for implementation and robust to errors, but requires high energy consumption and latency due to dismissing temporal information in spike trains. The temporal coding, like Time to First Spike (TTFS) [65] and Inter-Spike Interval (ISI) [66], uses the timing of spikes to carry information. The spiking ratio of temporal coding depends on the time resolution: the higher time resolution requires more time steps but less spiking ratio. The time resolution may differ with different workloads, and the input spiking can vary from dense to sparse. Some sensors may generate temporal coding directly, like DVS: each pixel of DVS generates events asynchronously when it detects changes and naturally encodes the information with temporal coding. It is also common to use temporal rate to encode 1D physiological signal processing[67]. Compared with rate code, temporal code has been proved to provide higher information capacity with lower response times but requires higher timing precision[68].

Synapses state update combining: Event-driven systems like DVS usually encode events with Address Event Representation (AER), which is efficient for temporal coding. To make HyperSpikeASIC compatible with both the EMG and DVS datasets, we represent an EMG signal preprocessed by sigma-delta modulation with AER as well. The AER packet includes weight w, height h, channel c, and spiking time t. As in a real-time system, the input data is sorted by time t. The time resolution Δ of SNN does not need to be as high as the original data so that several consecutive input events may be bucketed in one time step. The input FIFO would combine the events that occurred in the same time step and store the spike time t in front of the events' address which fire from t to $t+\Delta$. In this case, there is no guarantee that the input spike is ordered by position in one time step as it is in the original input.

As the input data are event-driven, not all neuron or synapse states are required to update at each time step. Figure 7 shows the spiking and neuron state changes in 2 successive CONV layers. Each neuron can be represented with a letter and a number. The capital letter is used for the pre-synaptic neuron while the lowercase letter is for the post-synaptic neuron. For example, A2 represents the pre-synaptic neuron located at the second row first column. The arrows are the spikes and the shadows are neurons that need an update. If the Chebyshev distance between two fired neurons is smaller than the CONV kernel size K_{CONV}, i.e., B6, D5, some of the post-synaptic neurons, i.e., c5, c6, would be influenced by both.

We use a bitwise mask to store the addresses of neurons that



(a) Dense Input (b) Sparse Input

Fig. 7. 2 Layer Neurons with Convolution Linked Synapses

receive no spike in one time step. As the position of the post-synaptic whose state requires an update is the same between different output channels, the bitwise mask can be shared by all post-synaptic neuron channels: when the synapses state update PE receives one input, the mask bits of its position and its nearby $K_{CONV}^2 - 1$ position will be set. As shown in Figure 7, the mask bits corresponding to the shadowed neuron are set. The intermediate results of synaptic potential would store in the synapse state scratch-pad memory.

Synapses state readout: When the timestamp of the next input is $t+1$, which means all the necessary synaptic states have been updated, the synapses state update PEs are in synapses state readout stage of in Figure 6: transmit the non-zero state to buffer and clear the scratch-pad memory. Read the mask byte by byte and translate the multi-hot position to serials of neuron state address. In this approach, only the updated synaptic state will be read and the non-updated position will be skipped. Similar to the input FIFO, the buffers between synapses and neuron PEs use a time stamp as a delimiter between two time steps.

The synapses state buffer decouples the synapses state update and neuron state update: all the synapses PE work at the same time step while the neuron PE may work at a different time step. Because the cycles required to process one-time-step data of synapse and neuron state update may be different, which depends on the number of spikes of pre-synaptic neuron and the number of overlapped post-synaptic neurons respectively, the bottleneck may be changed even during processing the same sample. Therefore, there would be some bubbles like the one between synapses state readout and neuron state update shown in the normal mode of Figure 6. From PE's perspective, we may reduce the processing time of neuron state update by adding more neuron PEs, thus reducing the possibility of bubbles. However, more neuron PEs means a larger area and more likely to be idle. Therefore, we optimized the ratio of the number of synapses PEs to the number of neuron PEs via $\beta = N_{SYN}/N_{NEU}$ to make the tradeoff between throughput and area, which is discussed in Section V-C. The number of FIFOs in the synapse state buffer equals the number of synapses PEs. As $N_{SYN} \neq N_{NEU}$, a neuron PE may consume data stored in different FIFO.

B. Neuron State Update

Because the update is event-driven, the neurons that receive no spikes are skipped. Therefore, there may be multi-time step intervals Δt between one neuron update and the next update. Because there is no input spike between timestamp t and $t + \Delta t$, we have:

$$S_{in}^{\tau} = 0, \ \forall \ \tau \in (t, t + \Delta t) \tag{6}$$

Rewrite equation 1:

$$\begin{aligned} P_j^{t+\Delta t} &= \alpha_0^{\Delta t} P_j^t + S_{in}^{t+\Delta t}, \\ R_i^{t+\Delta t} &= \alpha_0^{\Delta t}(R_i^t + U_i^t S_i^t), \\ U_i^{t+\Delta t} &= \sum_j W_{ij} P_j^{t+\Delta t} = \alpha_0^{\Delta t} U_i^t + \sum_j W_{ij} S_{in}^{t+\Delta t}, \\ S_i^{t+\Delta t} &= \Theta(U_i^{t+\Delta t} - R_i^{t+\Delta t}). \end{aligned} \tag{7}$$

Following (7), the neuron state memory stores neuron potential U_i^t, refractory R_i^t and last fire timestamp. As shown in neuron state update stage of Figure 6, the accumulated synapses state $\sum_j W_{ij} S_{in}^{t+\Delta t}$ is added to to decayed neuron potential $\alpha_0^{\Delta t} U_i^t$. If the difference between potential and refractory exceeds the threshold, the neuron would fire and update its last fire timestamp. The exponential calculation unit (marked as Exp Uint) and two multipliers in the neuron state update PE of Figure 4 are used for decay.

The exponential calculation: The exponential calculation unit is shown in Figure 8. It is based on a piecewise lookup table and has two modes: fast mode for neuron state update and precise mode for neuron state readout. The first part of the lookup table stores the key as an arithmetic sequence and the common difference is 1. The common difference between the key in the second part is the maximum key in the first part. The set of keys of two parts of the lookup table is $\{x | x < 2^{M_1}, x \in \mathbb{N}^+\}$ and $\{2^{(x+M_1)} | x < 2^{M-M_1}, x \in \mathbb{Q}\}$ respectively, where M is the input width of the exponential calculation unit and M_1 is the width of the first part of the lookup table. We choose $M_1 = 5, M = 8$ for Figure 8. In the fast mode, we use the fetched value as the output of the exponential calculation unit. If the most significant $M - M_1$ bits is zero, fetch from the first part, otherwise fetch from the second part. Because it's rare that one neuron doesn't receive any spike for a long time during SNN inference, we would fetch most of the results from the first part without precision

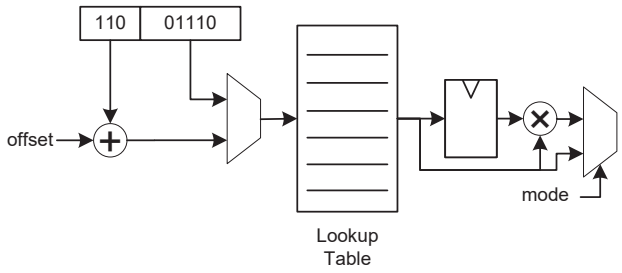


Fig. 8. Exponential Unit of Neuron PE

loss. We are going to readout the final neuron potential and feed them to HDC. Because some neurons may be inactive for a long time, but we expect a more precise output to feed to the HDC classifier, the exponential unit will work at the precise mode to interpolate. Because $\alpha_0^{x_1+x_2} = \alpha_0^{x_1}\alpha_0^{x_2}, x_1 = x\%2^{M_1}, x_2 = \lfloor x/2^{M_1} \rfloor$, the exponential unit multiplies the fetched result from the first part and the second part to get the precise result.

C. Synapses Update Bypass

When the input is sparse, there are only a few input spikes in one time step. As shown in Figure 7.b, it is inefficient to integrate the synapse state because there are few overlapping post-synaptic neurons. In such a case, one synapse PE might be able to produce enough intermediate results for one neuron PE. The synapse PE only fetches the weight and passes it to neuron PE directly. Figure 6 shows the pipeline of process data spike by spike without combining. There is no synapses state readout stage. As one post-synaptic neuron may receive multiple spikes in a time step, it will update its state multiple times with $\Delta t = 0$, indicating that potential and refractory decay operations have been done. When input events are pushed into the input FIFO, a counter will track the number of events that belong to the same time step. If the number of events that belong to the same time step exceeds a given threshold, indicating the input is sparse, the synapse update will bypass.

D. HDC architecture

The HDC encodes (maps) the readout neuron states to hypervectors (HVs) by a bank of multiplexers and an adder tree. The base vectors are bipolar, and thus the projection operation can be simplified to add or minus the readout features. As shown in the HDC part of Figure 4, we use a 2-1 multiplexer controlled by base vector $\mathbf{P}$ to select add or minus the readout features. After the adder tree, we detect the sign of each accumulated result as the encoded HV. Due to the limitation of readout bandwidth, we need multiple cycles to readout all features. Therefore, a register is inserted between the adder tree and the sign function. As the HV dimensions are large, it is inefficient to encode it all at once. We split each HV into several segments and handle them one at a time. During the training stage, we only need to accumulate the created HVs to the corresponding class HV. During the inference, we use Hamming distance for the similarity checking. As the HV is binary, we only need the sign bits of class HV. We xor the HVs with the class HVs' sign bits and accumulated results for hamming distance. Finally, we chose the class with the closest Hamming distance as the result.

V. Evaluation

A. Experimental Setup

HyperSpikeASIC software and hardware: We tested HyperSpike accuracy with and without bit errors on an Intel Core i7 7600 CPU using an optimized C++ implementation. HyperSpikeASIC has been implemented using SystemVerilog at the RTL level and synthesized by the Synopsys Design Compiler. The SRAM is generated by the ARM memory compiler with the 28nm node. We get the power information from the report of Synopsys Power Compiler. We evaluate the average energy by multiplying the average power and average running time. The cycle used for training is the same as inference, and thus the evaluated energy of training is equal to that of inference.

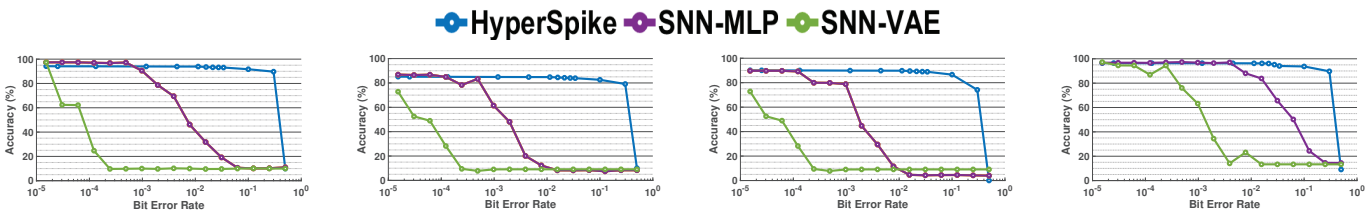
State of the art hardware comparison: We compare HyperSpikeASIC to HyperSpike [26] and state-of-the-art SNNs on neuromorphic hardware [35], [36], [38], [2]. HyperSpike accelerator was implemented with 56 Loihi cores for SNN by using the Intel Nahuku board [28] and HDC base on tinyHD [14] on N-MNIST, DVS-Gesture, and DVS-ASL as in work [26]. Besides, we expand the evaluation of the EMG dataset with the same setups as in [26]. The HDC in HyperSpike is implemented in SystemVerilog at the RTL level and synthesized by Synopsys Design Compiler with the 45 nm open-source NanGate cell library [31]. The results of HyperSpike are scaled down adopting CACTI for memory area and power, and the scaling trend of Intel [70] for the logic cells area. The power of logic cells is first obtained by the report of Synopsys Power Compiler and then scaled using HSPICE simulations with Predictive Technology Model (PTM) [71].

State of the art software comparison:We implemented the SNN models with PyTorch and HD modules in Python. The trained SNN model and VAE is implemented with DECOLLE[7]. The configuration of SNN and HDC is shown in Table II. The SNN-MLP is tested with at most 3 layers. When we test the HyperSpike, the HDC is directly connected to layer 1. The SNN-MLP will also take the layer 2 and layer 3. The HyperSpike is updated step by step. The number of time step used for each dataset is also list in Table II

Benchmarks: We tested our proposed approach on 3 event-based DVS datasets and using Electromyography (EMG) data. The DVS datasets are: 1) the Neuromorphic MNIST(N-MNIST) Handwritten digit Recognition dataset [72]; 2) the IBM DvsGesture dataset [58]; and 3) the American Sign Language (ASL) DVS dataset [73]. The N-MNIST dataset consists of 32×32, 300ms long event data streams of MNIST images recorded with an ATIS Camera [57]. The dataset contains 60,000 training event streams and 10,000 test event streams. The IBM DVSGesture dataset consists of recordings of 29 different individuals performing 10 different gestures, such as clapping, and an 'other' gesture class containing gestures that do not fit into the first 10 classes [58]. The DVS-ASL dataset contains 24 classes corresponding to letters A-Y, excluding J, in American Sign Language recorded using a DAVIS 240C event-based sensor [59]. The dataset contains 4200 240×180 100ms long event data streams of each letter, for a total of 100,800 samples.

TABLE I
ACCURACY OF HyperSpike VS OTHER SNN CLASSIFIERS AT FULL PRECISION AND QUANTIZED TO 16 BITS. PERFORMANCE NUMBERS ARE RELATIVE TO QUANTIZED SNN+MLP

Dataset	SNN+MLP [27]	Quantized SNN+MLP [60]	SNN+VAE [69]	Quantized SNN+VAE [69]	HyperSpike	Quantized HyperSpike		
Metrics	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy	Speedup	Energy Efficiency
N-MNIST	98.4%	97.4%	98.2%	97.4%	98.6%	95.2%	1.1×	1.1×
DVS-Gesture	93.8%	86.8%	83.6%	72.8%	87.2%	85.3%	14×	2.2×
DVS-ASL	89.7%	87.1%	89.7%	87.1%	88.2%	87.8%	14.9×	4.6×
EMG	97.2%	96.8%	95.8%	97.3%	97.6%	96.6%	1.1×	1.5×



(a) N-MNIST (b) DVS-Gesture (c) DVS-ASL (d) EMG

Fig. 9. Impact of Varying Levels of Bit Error Rates on the Accuracy of Quantized HyperSpike vs Other Quantized SNN Models.

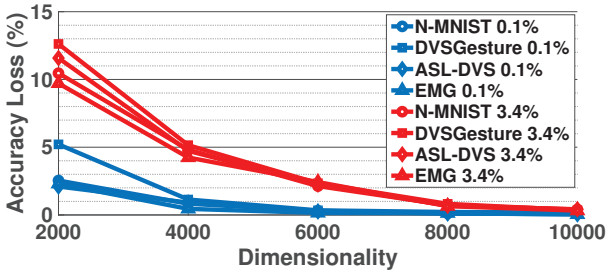


Fig. 10. Impact of Dimensionality on the Robustness of HyperSpike

TABLE II
CONFIGURATION OF SNN

Dataset	Input Size	Layer1	HDC Dimession	Layer2	Layer3	#Time Steps
N-MNIST	32×32	CONV(7×7 × 16)	4096	CONV(7×7 × 32)	CONV(7×7 × 32)	50
DVS-Gesture	32×32	CONV(7×7 × 32)	4096	CONV(7×7 × 64)	CONV(7×7 × 64)	100
DVS-ASL	60×30	CONV(7×7 × 16)	4096	CONV(7×7 × 32)	CONV(7×7 × 32)	100
EMG	64×1	FC(128)	4096	FC(64)	FC(32)	100

In addition to the event-based datasets, we also evaluate HyperSpikeASIC on Electromyography (EMG) data. The data is collected from an array of 64 electrodes connected to the forearm at a sampling frequency of 2000Hz. The dataset consists of 6 different hand and wrist gestures with 12 seconds of repetitions of each of the gestures. The data is converted into spikes using sigma-delta modulation before being input into SNN layers.

B. HyperSpike *Accuracy with Bit Errors*

Table I compares HyperSpike with SNNs using different classifiers at the output of the SNN network. The results show that HyperSpike with a full precision SNN is able to achieve similar accuracy to state-of-the-art solutions. HyperSpike is able to achieve accuracy within 2.6%(1%) of the other classifiers (quantized). However, HDC makes HyperSpike significantly more energy efficient (2.6x) because HyperSpike only needs to use the first layer of the SNN while its HDC layer uses parallel operations which are easily accelerated in hardware. HyperSpike is significantly more robust (58.3x) over the current state-of-the-art SNN classifiers. Figure 9 shows a comparison of HyperSpike with SNNs using different classifiers at the output of the SNN network. All SNN networks are quantized to 16-bit fixed-point precision [60] to reduce the impact of the bit errors. If an exponent bit is flipped in a floating point number, the resulting numerical error is more significant than flipping a mantissa bit. Taking this further, binary representations offer the highest resilience to bit flip errors as no matter which bit is flipped, the resulting numerical error is the same. This gives HyperSpike an advantage over SNN classifiers as the HDC classification portion of HyperSpike uses binary quantization. Our experiments show that HyperSpike is $31.4\times$ more robust than SNNs using MLPs (lost $31.4\times$ less accuracy than SNNs using MLPs) as a classifier when the BER is 0.1%, which is a typical error rate for systems that have high bit errors. HyperSpike is $58.3\times$ more robust than SNNs using MLPs as a classifier when the BER is 3.4%. We additionally tested SNNs using a VAE for classification.Our results apply to HyperSpike running on a ReRAM architecture such as [74]. They can similarly also extend to wireless communication errors if the parameters are sent and shared across different devices in a network. This indicates that one could create a computing architecture for HyperSpike that utilizes BER emerging hardware and does not need to add the overhead of error correction as HyperSpike is robust to the errors due to its HDC layer. Figure 10 tests HyperSpike with varying dimensionalities in typical (blue lines - 0.1%) and high (red lines - 3.4%) bit error rates. The results indicate that HyperSpike robustness scales with the dimensionality of the HDC model used for classification. As we increase the dimensionality of the HDC model, the accuracy loss decreases. This is consistent with prior work, which has shown that HDCs robustness is a function of high dimensionality [25].

C. SNN Configurations

To better understand the trade-off between area and throughput, we fix the number of neuron PE $N_{NEU} = 16$ and change the ratio to the number of synapse PEs via $\beta = N_{SYN}/N_{NEU}$.

TABLE III
COMPARISON OF THE RESULTS WITH THE SPIKING NEURAL NETWORK IMPLEMENTATIONS

	BWSNN [36]	AsycnSNN[35]	LSMCore [2]	DVS-Gestrue[38]	This Work			
Technology	90nm	28nm	40nm	28nm (TrueNorth[75])	28nm			
Area	2.07mm^2	1.28mm^2	18.49mm^2	4.3cm^2	2.70mm^2			
Power	0.62mW	3.42mW	4.9W	178.8 mW	480.1mW			
Frequency	10MHz	6.7MHz	400MHz	-	500MHz			
Neurons	8K	1280	1.31M	4K	32K			
Dataset	**MNIST**	**N-MNIST**	**N-MNIST**	**DVS-Gestrue**	**N-MNIST**	**DVS-Gestrue**	**DVS-ASL**	**EMG**
Net	BWSNN	MLP	LMS	SCNN	CONV+HDC	CONV+HDC	CONV+HDC	FC+HDC
Accuracy	98.0%	95.7%	98.7%	96.5%	95.2%	85.3%	87.8%	96.6%
Energy	0.59uJ/Image	3.97pJ/Synaptic Operation(SOP)	3.38mJ/Image	18.8mJ/Image	16.30uJ/Image	205.61uJ/Image	237.52uJ/Image	6.11uJ/read
Throughput	1.05K/s	-	1.44k/s	9.5/s	17.5K/s	2.18K/s	2.01k/s	78.4K/s

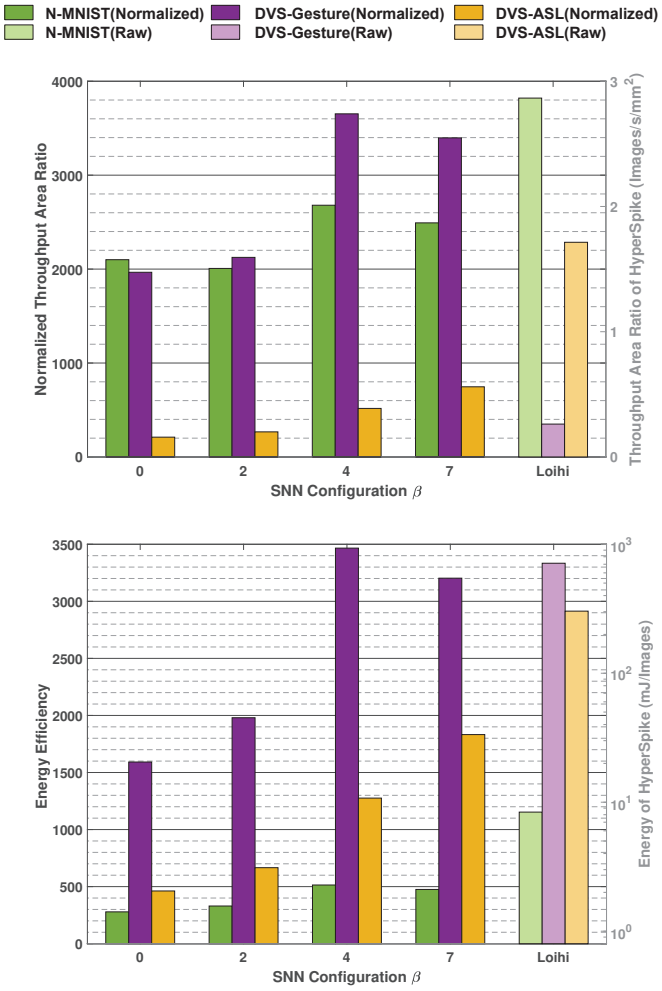


Fig. 11. Comparison of different β configuration of purposed SNN vs Intel's Loihi on Throughput Area Ration and Energy.

The throughput-to-area ratio and energy with different datasets and configuration β are shown in Figure 11. The results are normalized using those of the HyperSpike as the denominator. The raw results of the HyperSpike (marked as Loihi on the x-axis) are shown at the right of the figures with a lighter color. We chose $\beta = 0, 2, 4, 7, 49$, where $\beta = 0$ represents the area of synapses PEs and the synapses state memory described in IV.A is removed and it could only work without combining spikes; $\beta = 7$ represents the number of synapses neuron PEs is as much as the convolutional kernel size of SNN, as $_{CONV} = 7$. We get the largest boost on both throughput area ratio and energy efficiency when $\beta = 4$ on N-MNIST and DVS-gesture dataset but when $\beta = 7$ on DVS-ASL. This is because the workload of synapse PEs is heavier than neuron PEs. As we convert the input size of DVS-ASL from 240×180 to 60×30 by combining pixels, the number of neurons is reduced, but the number of input events remains. Because each neuron will update no more than once per time step, the workload of neuron PEs is reduced. Thus, for DVS-ASL dataset, we can get more benefits from increasing β.

We got the maximal throughput area ratio and energy efficiency boost on DVS-gesture. This is because the complexity of connectivity among neurons in DVS-gesture is the highest. The overhead of routing in Loihi is not trivial. However, thanks to the customized design, we can determine the data transfer path in hardware. We got the minimal throughput area ratio boost on ASL but the minimal energy efficiency boost on N-MNIST. That is because the input of N-MNIST is much more sparse than the input of ASL. Plenty of time is wasted on communication between Loihi and the host on N-MNIST and the chip is idle, which consumes less energy. The overhead of inter-chip communication is more significant for N-MNIST than DVS-ASL. In the proposed architecture, the generated feature could be consumed by the HDC directly. However, as the number of neuron PEs is less than Loihi when many neuron states need to update, we can only achieve less throughput boost.

Therefore, we take $\beta = 4$ to trade off performance and overhead. HyperSpikeASIC contains 16 neuron PEs and 64 synapses PEs. The width of the SNN weights, which are randomly initialized, and the width of synapse states is 16. Each neuron PE is able to store 2048 neuron states and each state is 45-bit width (16 bits for potential, 16 bits for refractory, and the remaining 13 bits for last fire time). The exponent lookup table is $16bits \times 128$, 96 of which are used for the first part of linearly growing keys and 32 of which are used for the second part of geometrically growing keys.

D. HyperSpike *Accelerator Performance and Statistics*

Table III summarizes the comparison of results among different implementations and performance in different datasets. Compared with previous SNN approaches, we achieve higher throughput on DVS-Gestrue and N-MNIST. However, we consume more energy on handwritten digit recognition tasks compared with [36]. One major reason is that work uses a standard MNIST dataset instead of DVS output and thus requires fewer spikes. In the [35], energy efficiency is normalized to synapse operation (SOP). It adopted an asynchronous design

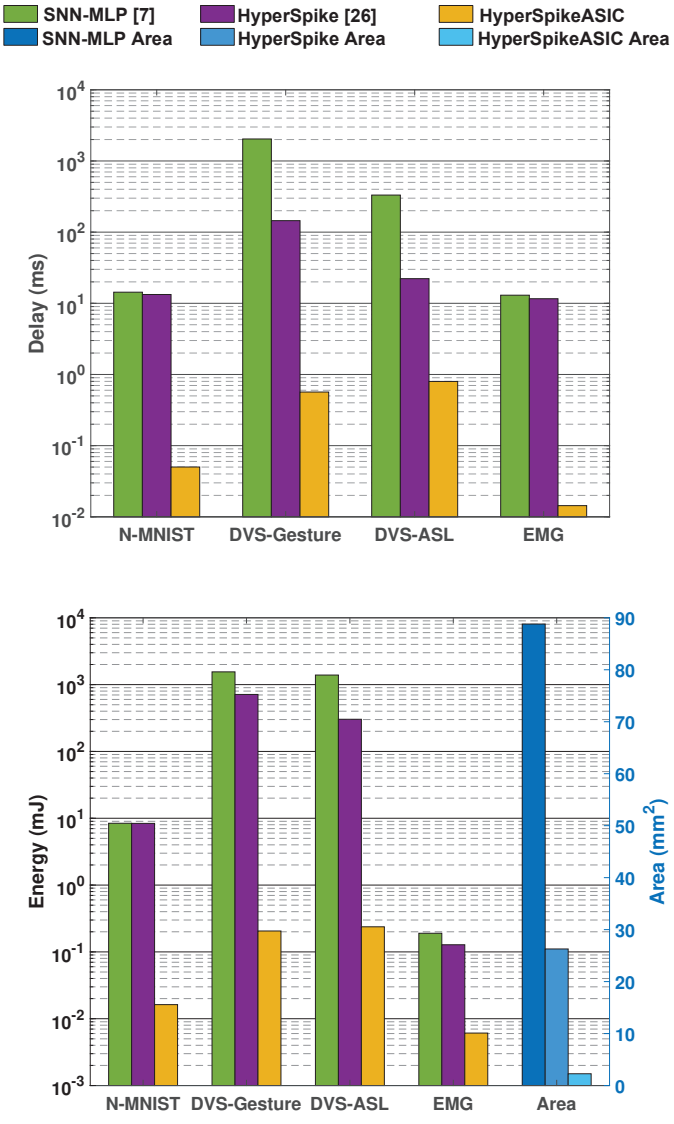


Fig. 12. Performance, Energy Consumption and Area of HyperSpikeASIC as compared to HyperSpike [26] and SNN[7] on Intel's Loihi[1]

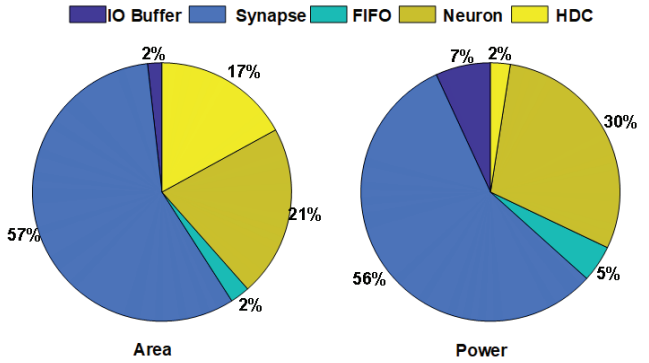


Fig. 13. Area and Power Breakdown

to increase the energy efficiency of handling each synaptic operation. But it must update all the synapses and neurons for each input, while HyperSpikeASIC will skip the inactive ones. As the HyperSpikeASIC can process 64 synapse state per cycle, the energy consumption will be $\frac{480.1mw}{500MHz \times 64SOP/cycle} = 15pJ/SOP$. This metric of TrueNorth is typically 26pJ/SOP [75].

Figure 12 compares purposed HyperSpikeASIC with HyperSpike [26] and a more traditional SNNs[7] using an MLP as the output classifier running on an Intel Loihi chips for acceleration. The results show that HyperSpikeASIC is $7,549\times$ and $3,467\times$ more energy-efficient and $4,437\times$ and $656\times$ faster than traditional SNN MLP and HyperSpike respectively. There are two main reasons for this. (1) Only one SNN layer is required by HyperSpike model and the HDC layer is more efficient and faster than the traditional SNN MLP classifier. Since SNNs dominate both latency and energy consumption of the whole design, the algorithm that uses simple and highly paralleled HDC layers instead of MLP SNN and compresses feature extraction to only one layer is more efficient. (2) The proposed ASIC design reduces the communication overhead between SNN and HDC and has been optimized for event-based classification tasks such as those presented in this paper.

We can see that the difference in performance between SNN MLP and HyperSpike [26] in the delay and energy on the N-MNIST and EMG is trivial because the network scale of these two datasets is small. HyperSpike boosts the performance mainly by shrinking the SNN, which means the smaller the original network, the less performance gain HyperSpike provides. However, the HyperSpikeASIC reduces the delay significantly for these two datasets compared with HyperSpike because inter-chip communication overhead accounts for more of the delay. We see the largest energy savings on the DVS-Gesture and DVS-ASL when comparing HyperSpikeASIC with HyperSpike [26]. This is because the customized design reduces the inner-chip communication overhead and is more energy-efficient on state updates. The differences between datasets are largely due to the differences in the network architectures, with larger architectures requiring more time and energy to execute.

Another advantage of our accelerator over state-of-the-art is the reduced chip area. Figure 12 compares the area needed to accelerate HyperSpike with the area needed to accelerate a traditional SNN with an Intel Loihi chip. For the traditional SNN, we only include the percent area of the Loihi chip used for acceleration. For HyperSpike, we add up the area from the number of Loihi cores our chip needs plus the area needed for our HDC ASIC. For HyperSpikeASIC, the area takes both SNN and HDC into account. We compare the chip area needed for the largest dataset, DVSGesture. As a result, this hardware could run all three datasets. The data shows that HyperSpikeASIC is $32.88\times$ and $9.73\times$ smaller than the SNN-MLP accelerator and HyperSpike, respectively.

Figure 13 shows the area and power breakdown of different components. The synapses state update takes the majority of area and power consumption. There are 4 times more synapse PEs than neuron PEs, but the area and power consumption of synapse PEs are around two times more than neuron PEs; each synapse PE is around 2 times larger and has greater energy consumption than neuron PE.

E. Overhead

The throughput bottleneck of HyperSpikeASIC is SNN. More specifically, because of how slow the SNN is, the HDC will be idle 80% of the time on average. The SNN also takes up

the majority of the area - 83%. Because the SNN needs random access to the neuron state and synapse state, the throughput of the SNN is limited by the memory bandwidth of state storage. As a result, we use multiple memory banks to provide sufficient memory bandwidth. The state storage consumes 88% area of the SNN module.

VI. Conclusion

In this paper, we create HyperSpike, a method for improving the energy efficiency and robustness of spike neural networks with Brain-Inspired Hyper-Dimensional Computing. It expands the HDC's applications to event-based data. The first layer of HyperSpike is a randomly initialized SNN layer that does not need to be trained. This layer processes the event-based signal data from a neuromorphic sensor and outputs feature vectors. Then, the trained HDC layer interprets these feature vectors to perform classification. By combining SNNs and HDC in this way, HyperSpike is able to achieve high classification accuracy with HDC on event-based data while being much smaller, faster, and more energy efficient. We show that HyperSpike is able to process not only original event-based data like the output of dynamic vision sensors but physiological signals like electromyography with sigma-delta modulation, which shows the possibility of cascading HyperSpike to traditional ADC instead of just event-based sensors. Our results show that HyperSpike is $31.4\times$ more robust to errors than traditional SNNs. Our HW implementation HyperSpikeASIC is $601\times$ faster and $3.467\times$ more energy efficient than the hardware implementation of HyperSpike with general purpose SNN acceleration platform Loihi[1] and tinyHD[14]. and $12.2\times$ faster and $211\times$ more energy efficient than the state-of-the-art SNN implementation[2].

References

[1] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

[2] L. Wang, Z. Yang, S. Guo, L. Qu, X. Zhang, Z. Kang, and W. Xu, "Lsmcore: Ieeebiographya 69k-synapse/mm^2 single-core digital neuromorphic processor for liquid state machine," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2022.

[3] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci, "Online few-shot gesture learning on a neuromorphic processor," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 512–521, Oct 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9229141

[4] J. Park, J. Lee, and D. Jeon, "7.6 a 65nm 236.5nj/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 140–142.

[5] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, B. Kay *et al.*, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.

[6] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural Comput.*, vol. 30, no. 6, p. 1514–1541, jun 2018. [Online]. Available: https://doi.org/10.1162/neco_a_01086

[7] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.

[8] C. Lee, P. Panda, G. Srinivasan, and K. Roy, "Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning," *Frontiers in neuroscience*, vol. 12, p. 435, 2018.

[9] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike-based visual categorization using reward-modulated stdp," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 6178–6190, 2018.

[10] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis *et al.*, "Event-based vision: A survey," *arXiv preprint arXiv:1904.08405*, 2019.

[11] S. Sheik, S. Paul, C. Augustine, C. Kothapalli, M. M. Khellah, G. Cauwenberghs, and E. Neftci, "Synaptic sampling in hardware spiking neural networks," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 2090–2093.

[12] R. Graca and T. Delbrück, "Unraveling the paradox of intensity-dependent dvs pixel noise," *ArXiv*, vol. abs/2109.08640, 2021.

[13] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[14] B. Khaleghi, H. Xu, J. Morris, and T. Š. Rosing, "tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 408–413.

[15] A. Dutta, S. Gupta, B. Khaleghi, R. Chandrasekaran, W. Xu, and T. Rosing, "Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, ser. GLSVLSI '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 281–286. [Online]. Available: https://doi.org/10.1145/3526241.3530331

[16] M. Imani, Z. Zou, S. Bosch, S. A. Rao, S. Salamat, V. Kumar, Y. Kim, and T. Rosing, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 221–234.

[17] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2015.

[18] M. Imani *et al.*, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[19] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 356–371.

[20] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. Rosing, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1591–1594.

[21] Y. Guo, M. Imani, J. Kang, S. Salamat, J. Morris, B. Aksanli, Y. Kim, and T. Rosing, "Hyperrec: Efficient recommender systems with hyperdimensional computing," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 384–389.

[22] B. Khaleghi, M. Imani, and T. Rosing, "Prive-hd: Privacy-preserved hyperdimensional computing," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[23] Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, ser. DATE '20. San Jose, CA, USA: EDA Consortium, 2020, p. 115–120.

[24] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, 2017, pp. 1–8.

[25] J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, and T. Rosing, "Hydrea: Towards more robust and efficient machine learning systems with hyperdimensional computing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 723–728.

[26] J. Morris, H. W. Lui, K. Stewart, B. Khaleghi, A. Thomas, T. Marback, B. Aksanli, E. Neftci, and T. Rosing, "Hyperspike: Hyperdimensional computing for more efficient and robust spiking neural networks," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 664–669.

[27] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2020.00424

[28] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, P. Joshi, A. Lines, A. Wild, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. PP, no. 99, pp. 1–1, 2018.

[29] W. Gerstner and W. Kistler, *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[30] F. Zenke and S. Ganguli, "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks," *Neural Computation*, vol. 30, no. 6, pp. 1514–1541, 06 2018. [Online]. Available: https://doi.org/10.1162/neco_a_01086

[31] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1m-synapse 3.8-pj/sop spiking neural network with on-chip stdp learning and sparse weights in 10-nm finfet cmos," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, 2019.

[32] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B.-D. Rubin, F. Akopyan, E. McQuinn, W. P. Risk, and D. S. Modha, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–10.

[33] H. Tang, H. Kim, H. Kim, and J. Park, "Spike counts based low complexity snn architecture with binary synapse," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1664–1677, 2019.

[34] M. Koo, G. Srinivasan, Y. Shim, and K. Roy, "sbsnn: Stochastic-bits enabled binary spiking neural network with on-chip learning for energy efficient neuromorphic computing at the edge," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2546–2555, 2020.

[35] J. Zhang, M. Liang, J. Wei, S. Wei, and H. Chen, "A 28nm configurable asynchronous snn accelerator with energy-efficient learning," in *2021 27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, 2021, pp. 34–39.

[36] P.-Y. Chuang, P.-Y. Tan, C.-W. Wu, and J.-M. Lu, "A 90nm 103.14 tops/w binary-weight spiking neural network cmos asic for real-time object classification," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[37] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch *et al.*, "From the cover: Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 113, no. 41, p. 11441, 2016.

[38] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7388–7397.

[39] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[40] L. Zhang, J. Yang, C. Shi, Y. Lin, W. He, X. Zhou, X. Yang, L. Liu, and N. Wu, "A cost-efficient high-speed vlsi architecture for spiking convolutional neural network inference using time-step binary spike maps," *Sensors*, vol. 21, no. 18, p. 6006, 2021.

[41] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 640m pixel/s 3.65mw sparse event-driven neuromorphic object recognition processor with on-chip learning," in *2015 Symposium on VLSI Circuits (VLSI Circuits)*, 2015, pp. C50–C51.

[42] P. Kanerva, "Encoding structure in boolean space," in *ICANN 98*. Springer, 1998, pp. 387–392.

[43] N. R. Shanbhag, N. Verma, Y. Kim, A. D. Patil, and L. R. Varshney, "Shannon-inspired statistical computing for the nanoscale era," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 90–107, 2018.

[44] A. Rahimi, A. Tchouprina, P. Kanerva, J. d. R. Millán, and J. M. Rabaey, "Hyperdimensional computing for blind and one-shot classification of eeg error-related potentials," *Mobile Networks and Applications*, pp. 1–12, 2017.

[45] M. Imani *et al.*, "A binary learning framework for hyperdimensional computing," in *DATE*. IEEE/ACM, 2019.

[46] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 445–456.

[47] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*. ACM, 2019, pp. 53–62.

[48] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[49] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 64–69.

[50] H. Li *et al.*, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *IEDM*. IEEE, 2016, pp. 16–1.

[51] C. Eliasmith, T. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *Science*, vol. 338, no. 6111, pp. 1202–1205, 2012.

[52] E. P. Frady and F. T. Sommer, "Robust computation with rhythmic spike patterns," *Proceedings of the National Academy of Sciences*, vol. 116, no. 36, pp. 18 050–18 059, 2019. [Online]. Available: https://www.pnas.org/content/116/36/18050

[53] E. P. Frady, D. Kleyko, and F. T. Sommer, "Variable binding for sparse distributed representations: Theory and applications," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2021.

[54] D. Kleyko, M. Davies, E. P. Frady, P. Kanerva, S. J. Kent, B. A. Olshausen, E. Osipov, J. M. Rabaey, D. A. Rachkovskij, A. Rahimi, and F. T. Sommer, "Vector symbolic architectures as a computing framework for nanoscale hardware," 2021.

[55] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[56] M. Davies, "Lessons from loihi: Progress in neuromorphic computing," in *2021 Symposium on VLSI Circuits*, 2021, pp. 1–2.

[57] C. Posch, D. Matolin, and R. Wohlgenannt, "A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 259–275, jan. 2011.

[58] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[59] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240$\times$ 180 130 db 3 μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014.

[60] H. W. Lui and E. Neftci, "Hessian aware quantization of spiking neural networks," 2021.

[61] K. H. Srivastava, C. M. Holmes, M. Vellema, A. R. Pack, C. P. H. Elemans, I. Nemenman, and S. J. Sober, "Motor control by precisely timed spike patterns," *Proceedings of the National Academy of Sciences*, vol. 114, no. 5, pp. 1171–1176, 2017. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1611734114

[62] T. Bu, W. Fang, J. Ding, P. Dai, Z. Yu, and T. Huang, "Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks," in *International Conference on Learning Representations*, 2021.

[63] B. Han, G. Srinivasan, and K. Roy, "Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 13 558–13 567.

[64] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[65] B. Rueckauer and S.-C. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[66] R. Snider, J. Kabara, B. Roig, and A. Bonds, "Burst firing and modulation of functional connectivity in cat striate cortex," *Journal of Neurophysiology*, vol. 80, no. 2, pp. 730–744, 1998.

[67] N. Nuntalid, K. Dhoble, and N. Kasabov, "Eeg classification with bsa spike encoding algorithm and evolving probabilistic spiking neural network," in *International conference on neural information processing*. Springer, 2011, pp. 451–460.

[68] D. Auge, J. Hille, E. Mueller, and A. Knoll, "A survey of encoding techniques for signal processing in spiking neural networks," *Neural Processing Letters*, vol. 53, no. 6, pp. 4693–4710, 2021.

[69] K. Stewart, A. Danielescu, L. Supic, T. Shea, and E. Neftci, "Gesture similarity analysis on event data using a hybrid guided variational auto encoder," *arXiv preprint arXiv:2104.00165*, 2021.

[70] M. T. Bohr and I. A. Young, "Cmos scaling trends and beyond," *IEEE Micro*, vol. 37, no. 6, pp. 20–29, 2017.

[71] "Predictive technology model," http://ptm.asu.edu/.

[72] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, nov 2015.

[73] Y. Bi, A. Chadha, A. Abbas, , E. Bourtsoulatze, and Y. Andreopoulos, "Graph-based object classification for neuromorphic vision sensing," in *2019 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2019.

[74] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 802–815.

[75] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1254642

Tianqi Zhang is currently pursing the MS degree with the Department of Electrical and Computer Engineering, University of California at San Diego, CA, USA. His current research interests include algorithm and hardware co-design, neuromorphic computing, and domain-specific accelerators.

Justin Morris received his Ph.D. degree from University of California, San Diego and San Diego State University in 2022. He is an Assistant Professor of Computer Engineering at the California State University, San Marcos. His research interests include hyperdimensional computing, machine learning, and processing in-memory.

Kenneth Stewart (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the University of California Irvine. His current research focuses on developing learning algorithms for neuromorphic hardware and their application to areas such as computer vision and robotics. His research interests include neuromorphic computing, online learning, robotics, artificial intelligence, and applications thereof.

Hin Wai Lui received has B.A in Engineering from Cambridge University and MPhil in Engineering and Entrepreneurship from the Hong Kong University of Science and Technology. He is currently pursuing a PhD in computer science at UC Irvine. His main interests are spiking neural network training, neuromorphic hardware, and brain-inspired computing.

Behnam Khaleghi received the BS and MS degrees from the Department of Computer Engineering, Sharif University of Technology, in 2013 and 2016, respectively. He is currently working toward the PhD degree with the Department of Computer Science and Engineering, University of California San Diego, CA, USA. His research interests include brain-inspired computing, ML acceleration, reconfigurable computing, and VLSI design automation.

Anthony Thomas received his Bachelor of Science, with high honors, in Agricultural Economics from the University of California, Berkeley in 2013 and previously worked as a Senior Research Analyst at Brown University for Justine Hastings and Jesse Shapiro. He is currently a graduate student in Computer Science and Engineering at the University of California, San Diego. His research interests include neurally plausible models of data representation and learning.

Thiago Goncalves-Marback is currently pursuing the MS degree with the Department of Computer Science and Engineering, University of California San Diego, CA, USA. His research interests include hyperdimensional computing and machine learning,

Baris Aksanli (Member, IEEE) is currently an Assistant Professor with the Electrical and Computer Engineering Department, San Diego State University, San Diego, CA, USA. Previously, he was a Postdoctoral Researcher at the Computer Science and Engineering Department, University of California San Diego. As a Researcher, his affiliations include the Multi Scale Systems Center (MuSyC), the TerraSwarm Research Center, and the Center for Networked Systems (CNS); and the collaborators of his projects include Google, Microsoft, Panasonic, Intel, and IBM. His research interests include energy efficiency and peak power management of large-scale systems, such as data centers and smart grids, efficient battery usage in data centers and residential houses, battery lifetime modeling, cost and energy aware automation of residential houses, learning techniques to enhance user behavior modeling and context extraction, house/building/data center, and grid interaction.

Emre O. Neftci (Member, IEEE) received the M.Sc. degree in physics from Ecole Polytechnique Federale de Lausanne, Switzerland, and the Ph.D. degree from the Institute of Neuroinformatics, University of Zurich and ETH Zurich, in 2010. He is currently an Assistant Professor with the Department of Cognitive Sciences and Computer Science, University of California at Irvine. His current research explores the bridges between neuroscience and machine learning, with a focus on the theoretical and computational modeling of learning algorithms that are best suited to neuromorphic hardware and non-von Neumann computing architectures.

Tajana Rosing (Fellow, IEEE) received the MS degree in engineering management concurrently and the PhD degree from Stanford University, Stanford, CA, USA, in 2001. She is a professor, a holder of the Fratamico Endowed chair, and the director of System Energy Efficiency Laboratory, University of California at San Diego, La Jolla, CA. From 1998 to 2005, she was a full-time research scientist with HP Labs, Palo Alto, CA, while also leading research efforts with Stanford University, Stanford. She was a senior design engineer with Altera Corporation, San Jose, CA. She is leading a number of projects, including efforts funded by DARPA/SRC JUMP CRISP program with focus on design of accelerators for analysis of Big Data, DARPA and NSF funded projects on hyperdimensional computing, and SRC funded project on IoT system reliability and maintainability. Her current research interests include energy-efficient computing, cyber–physical, and distributed systems.