Superconducting Hyperdimensional Associative Memory Circuit for Scalable Machine Learning

Kylie Huch, Patricia Gonzalez-Guerrero, *Member, IEEE*, Darren Lyles, *Member, IEEE*, and George Michelogiannakis, *Senior Member, IEEE*

***Abstract*—We propose a generalized architecture for the first rapid-single-flux-quantum (RSFQ) a ssociative m emory circuit. The circuit employs hyperdimensional computing (HDC), a machine learning (ML) paradigm utilizing vectors with dimensionality in the thousands to represent information. HDC designs have small memory footprints, simple computations, and simple training algorithms compared to superconducting neural network accelerators (SNNAs), making them a better option for scalable SFQ machine learning (ML) solutions. The proposed superconducting HDC (SHDC) circuit uses entirely on-chip RSFQ memory which is tightly integrated with logic, operates at 33.3 GHz, is applicable to general ML tasks, and is manufacturable at practically useful scales given current SFQ fabrication limits. Tailored to a language recognition task, SHDC consists of ∼2-20M Josephson junctions (JJs) and is 10^2 (RSFQ) to $> 10^3$ (ERSFQ) times more energy efficient than an analogous CMOS HDC circuit including cooling and $> 10^4$ (RSFQ) to 10^6 (ERSFQ) times excluding cooling while achieving 78-84% higher throughput. SHDC is up to 10^7 (RSFQ) to 10^6 (ERSFQ) times more energy efficient t han t he s tate o f t he a rt R SFQ SNNA, SuperNPU, while achieving 62-99% higher throughput for all but the smallest NN accelerated by SuperNPU. To the best of the authors' knowledge, SHDC is currently the only superconducting ML approach feasible at practically useful scales for real-world ML tasks and capable of online learning.**

I. Introduction

With the slowdown of Moore's law and the end of Dennard scaling, superconducting digital computing offers a promising alternative for future high performance computing (HPC) systems due to its ability to operate at up to ∼100 GHz with low power dissipation [1], [2]. Of the variety of different superconducting digital logic families based on single-flux-quantum (SFQ) pulses, rapid-single-flux-quantum (RSFQ) logic is the most mature and remains the most common for high-speed circuit applications [3]. As such, we design and simulate a practical HDC circuit based on standard RSFQ logic gates [4].

Limited device density is one of the greatest challenges for superconducting digital computing currently, making area a critical constraint for SFQ circuits [1], [5]. Although RSFQ circuits with about one million Josephson junctions (JJs) [6], [7] and, more recently, close to ten million JJs [8] have been demonstrated, these chips had highly-regular shift register designs. In terms of complex logic and irregular RSFQ circuits, recently demonstrated chips have been limited to around 20-30 thousand JJs [9]. Additionally, cryogenic on-chip memory is widely regarded as a scarce resource [10]. These factors severely hinder the development of SFQ circuits for machine learning (ML) at practical scales due to the computational complexity and large memory footprints typical of these algorithms.

The authors are with the Computer Architecture Group, Applied Mathematics and Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, California, USA., Email: [kyliehuch,lg4er,dlyles,mihelog]@lbl.gov

A. Background and Related Work

At a high level, all ML approaches can be broken down into two basic stages: ***learning***, also referred to as training, and ***inference***, also referred to as classification. Typically, ML systems accept a single type of input data–for example: images, text, frequency data, etc.–and map it to some internally-used representation preserving the feature(s) of interest. During learning, the system constructs high-level class representations that capture the distinguishing statistical characteristics of the objects in that class either using explicitly labeled data in what is termed ***supervised learning***, or from naturally-arising similarities and differences among unlabeled data in what is termed ***unsupervised learning*** or ***clustering***. There are also other approaches using a combination of the aforementioned methods such as ***reinforcement learning*** and ***semi-supervised learning***. Most ML approaches can be used with any of the above methods. During classification, the system calculates the best matching class to unlabeled input data and returns the label of that class as the classification result. For additional details on ML methods and approaches please consult [11].

Learning is significantly more complex and time-consuming than classification. As such, many ML approaches perform training separately from classification in more powerful and less resource-constrained environments to enable deployment of the system on less complex hardware in what is termed ***offline*** or ***batch learning***. This approach comes at the expense of the robustness and adaptability of the system; once deployed, they cannot continue to learn, making such systems rigid, highly dependent on the training environment, and often susceptible to noise. Supervised learning methods typically necessitate offline learning. When the system is capable of continued learning after deployment, it is termed ***online*** or ***continuous learning***. The architecture used to deploy such systems must be capable of implementing the computations of both learning and classification; however, the resulting system benefits from greatly increased robustness and adaptability. Please consult [12] for additional details.

Neural networks (NNs) are the dominant ML approach currently and are capable of achieving near-human level accuracy for many tasks at the cost of memory, inference computational complexity, and training complexity [13], [14]. NNs consist of multiple layers each containing on the order of thousands of nodes connected in a predefined and fixed manner. Each node has tens to thousands of floating point parameters, resulting in millions to hundreds of millions of parameters for standard, modern NN architectures. Input data is mapped to typically float-valued ***activations*** of first-layer nodes which then propagate through the network based on the connectivity of the nodes and the values of the learned network parameters. Each node in the output layer represents a class and the one with the highest valued activation is the winner of the classification operation.

Classification in NNs involves on the order of millions of data-intensive matrix multiply-accumulate (MAC) operations per classification operation [13], [15]. Training involves both backwards and forwards passes over the data with the backward pass being far more computationally complex. Following the forward pass (classification), error is back-propagated through the entire network (backwards pass). This requires the repeated computation of extremely large and complex gradients followed by the adjustment of the many millions of learnable network parameters. In order for the network to successfully converge, this must be done over the entire training data set in many thousands of iterations, making NN training exceedingly energy and time intensive [13], [14], [16].

State-of-the-art superconducting NN accelerators (SNNAs) such as SuperNPU [15] use quantized network parameters both to reduce memory requirements and to better match the quantized nature of SFQ pulses. However, quantizing NN parameters to the low bitwidths ($\leq$8 bits) required for realistic implementation in SFQ technology inevitably introduces quantization error and reduces noise tolerance [17], [18]. Additionally, even with aggressively quantized parameters, the smallest NNs still require on the order of millions of parameters [19]–[24], necessitating the use of off-chip, non-SFQ memory as well as converters, resulting in increased latency and area [10]. Perhaps most importantly, due to the extreme complexity of NN training, SNNAs are designed to implement only the forward pass of NNs (classification). ***This makes SNNA architectures fundamentally incapable of online learning***. They must be initialized with pretrained, quantized network parameters and thus are unable to learn following deployment.

The matrix-based nature of NN computations results in a huge amount of data movement in the form of both network parameters and partial results. As such, NN accelerators require large amounts of on-chip memory in the form of buffers to ensure performance is not unduly limited by memory bandwidth [15]. This is especially important for SNNAs as the maximum operating frequency of the off-chip, non-SFQ memory–DRAM in the case of SuperNPU–is generally much slower than that of the SFQ portion of the circuit [2], [25]–[28].

While NNs excel in semiconductor-based computing environments, they are not a good match to superconducting digital environments due to the limitations on device density and lack of abundant cryogenic on-chip memory in conjunction with the extreme computational complexity, memory requirements, and training costs of NNs. Conversely, neuromorpic ML systems are uniquely well-suited for implementation in SFQ technology. A major barrier to the implementation of next-generation neuromorphic designs in traditional semiconductor technologies is their vast scales which result in large power consumption and data processing requirements [29]. As such, the implementation of these systems in superconducting environments has garnered increasing attention recently due to the remarkably high speed and low power consumption metrics of this technology, enabling the scaling of such systems beyond what is realistic in semiconductor technologies [29], [30]. Furthermore, the natural spiking behavior of JJs very closely resembles that of biological neurons; JJ critical current is analogous to the threshold potential of neurons, enabling biologically-plausible spike-based computing schemes [29]–[31]. [30] and [31] propose potential designs for such next-generation superconducting neuromoprhic designs capitalizing on the aforementioned properties and [29] provides a comprehensive review of cryogenic neuromorphic hardware.

In terms of currently feasible superconducting ML approaches, the only one the authors are aware of is the SFQ Discrete Hopfield Neural Network presented in [32]. This SFQ circuit is intended for image recognition tasks and is realistically fabricatable given current SFQ technology limits. However, the circuit is only capable of storing two, 8-bit memory patterns, making it unrealistic for use with real-world ML tasks at its current scale.

B. Summary

In this paper, we propose hyperdimensional computing (HDC) for efficient ML in superconducting digital computing. HDC is a Turing-complete neuromorphic computational paradigm in which computation is performed with vectors whose dimensionality is in the thousands, termed ***hypervectors*** [33]–[38]. Data is represented holographically within these hypervectors, meaning that data is distributed across the entire vector; no subset has any particular meaning [37], [39], [40]. Due to their hyperdimensional and holographic nature, the representations of HDC are extremely robust to noise and error from all sources [35], [38], [41]. The power of HDC as an ML approach lies in the topographical properties of its hyperdimensional (HD) representational space which is ideally suited for naturally expressing cognitive operations [33], [35], [38]. As such, the computations of HDC are quite simple, resulting in designs with small hardware footprints dominated by memory [34], [36]. Additionally, HDC is capable of learning classes from very little training data while achieving accuracy competitive with NNs [38], [40], [42], [43]. Furthermore, learning and classification are performed in the *same manner* in HDC, making these architectures capable of online learning without added computational overhead [36], [38]. See Table I for a glossary of HDC terms and Section II for additional details on HDC as an ML paradigm.

HDC's low computational complexity, small area footprints, simple algorithms, robustness, and fast, online learn-

TABLE I: Glossary of HDC terms.

Hypervector	Vector of length N where N=500-10k
HD Space	Representational space formed by the 2N possible binary hypervectors of length N
Projection Function	The mapping of the input data to the HD representational space
Seed Vector	Predefined hypervector chosen to represent a feature of the input data in HD space
Profile Vector	Hypervector representing input data in the HD representational space
Memory/Class Vector	Profile vector of a known class stored in item memory
Query Vector	Profile vector of an unknown class
Difference Vector	The element-wise XOR of two hypervectors ('1' at every element where values differ)
Encode Module (EM)	Projects input data to its profile vector (implements the projection function)
Associative Search Module (ASM)	Stores learned memory/class vectors, classifies query vectors
Item Memory Vector Node (IMVN)	Dedicated memory node within the ASM, holds one memory vector and calculates its Hamming distance from query vectors during classification
Seed Memory	Look-up table mapping all data features to their corresponding seed vectors
Item Memory	Learned memory/class vectors stored in the ASM

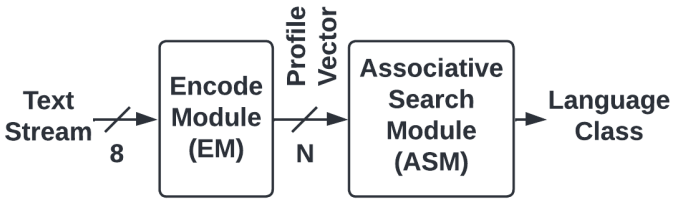

Fig. 1: Overview of HDC for language recognition.

ing capabilities make it ideally suited for implementing ML in SFQ technology.

In this paper, we present a generalized architecture for the first superconducting HDC (SHDC) design and evaluate it against an analogous CMOS HDC design [36] as well as the state-of-the-art SNNA, SuperNPU [15]. As we show, our SHDC architecture operates at 33.3 GHz and is 10^2 (RSFQ) to $> 10^3$ (ERSFQ) times more energy efficient than an analogous CMOS HDC circuit including cooling and $> 10^4$ (RSFQ) to 10^6 (ERSFQ) times excluding cooling while achieving 78-84% higher throughput, and is manufacturable at practically useful scales given current SFQ fabrication limits. SHDC is up to 10^7 (RSFQ) to 10^6 (ERSFQ) times more energy efficient than SuperNPU, while achieving 62-99% higher throughput for all but the smallest NN accelerated by SuperNPU (MobileNet).

To the best of the authors' knowledge, SHDC is currently the only superconducting ML approach that is feasible at practically useful scales for real-world ML tasks and capable of online learning. ***We argue that SHDC is a superior approach for implementing machine learning in superconducting digital computing.***

II. Hyperdimensional Computing (HDC)

In HDC, objects are represented in hyperdimensional (HD) space as N-dimensional vectors–termed ***hypervectors***–where N typically ranges from 500 to 10k [33], [35], [38], [40], [44]. Due to its high dimensionality, the topology of the representational space formed by these hypervectors is ideal for naturally expressing the types of cognitive operations required for artificial intelligence (AI), resulting in lightweight but powerful ML architectures [33], [36], [38], [45]. Higher-order class representations are formed by recoverably superimposing multiple object hypervectors of the same class into a single, averaged hypervector representing the class [36], [37]. These class hypervectors are the only things learned and stored in HDC. The natural, highly neuromorphic ability to compute in superposition underlies the remarkable error tolerance, fast learning capabilities, and robustness of solutions learned by HDC [33], [35], [38], [40], [42], [46].

Both NNs and HDC map data to a HD space to perform learning and classification; however, this happens differently in each paradigm. In NNs, the HD space–the parameter space of all nodes in the network–is used to perform the mapping of inputs to classes, represented by final layer nodes. NNs learn the correct "slice" of this HD space such that inputs are mapped to the desired classes. In HDC, the mapping is predefined and the HD representational space–the space of all possible hypervectors under a given representational scheme and dimensionality, N–is not constrained or stored. In this case, it is only the points within the space corresponding to class centroids which are learned and stored. Points are represented as individual hypervectors.

The training costs and memory footprints of NNs are so high because *the entire representational space* is learned and stored. By learning and storing *only the points in the representational space corresponding to classes*, HDC is able to solve the same ML tasks with far simpler hardware, smaller memory footprints, and less training data than NNs [38], [40], [42].

One highly desirable feature of HDC is one-shot learning: the ability to learn a class from one or very few passes over the training data [38], [40], [42], [47]. For example, an HDC algorithm achieved 97.8% classification accuracy in one pass over $\frac{1}{3}$ the training data required by the state-of-the-art support vector machine (SVM) on the same task [48].

Due to the exceptional accuracies of cutting-edge deep neural networks (DNNs) [21], [23], [24], it is extremely challenging to out-perform them in terms of accuracy. This is particularly true for image classification tasks in which feature extraction capabilities are crucial for learning transformation invariance [40], [46], [49]. To overcome these challenges, state-of-the-art HDC approaches for such tasks build feature-extracting kernels into their encoding schemes mapping input data to the HD representational space [50], [51]. HDC is capable of achieving comparable accuracies to DNN models on applications including computer vision, speech detection, robotics, and others. [40], [43], [49], [52]–[55].

We focus on the European language recognition task for our SHDC implementation presented in the following sections. The details of this task are discussed in Section III. The HDC algorithm implemented by our SHDC design achieves 96.7% accuracy on this task compared to the 97.9% accuracy achieved by a histogram-based nearest neighbor baseline classifier [36]. This algorithm makes use of binary hypervectors, the binary spatter code representational scheme, ∼1MB

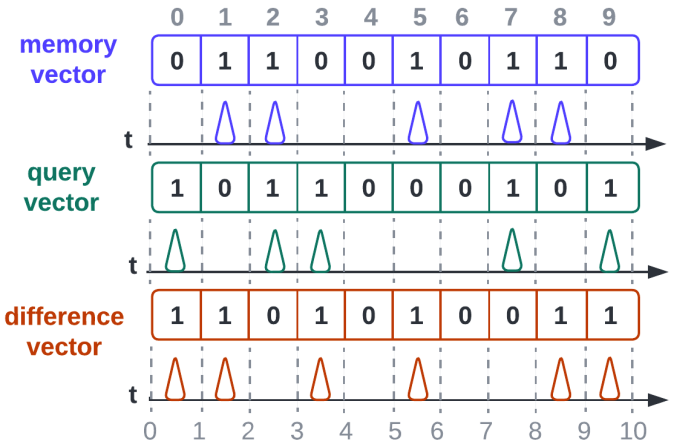

Fig. 2: Hypervector pulse stream representation.

training texts, ∼1kB evaluation texts, and generates language profiles based on consecutive groups of three characters (trigrams). It is discussed in detail in Sections III to V.

A high-level overview of a HDC ML architecture for language recognition is shown in Fig. 1. It is composed of two modules, the encode module (EM) and the associative search module (ASM). The EM projects input data to an N-dimensional hypervector in the representational space, termed the ***profile vector*** of the input data. During training when the class of the profile vector is known, it is written to one of the M dedicated item memory nodes in the ASM to form a new class. When the class of the profile vector is unknown, it is termed a ***query vector*** and searched for across the M classes stored in the ASM in a nearest-neighbor search.

There are many possible encoding schemes for HD vectors ranging from simple binary and bipolar representations to much more complex representations making use of floating point and imaginary numbers [37], [38], [40]. For simplicity, we use binary hypervectors and the binary spatter code [39] in our implementation following that of [36]. Fig. 2 shows how binary hypervectors are represented as pulse streams in our design. For binary hypervectors, ***Hamming distance*** is used as the similarity function. Hamming distance is a simple L1 distance measure consisting of the number of elements at which two vectors differ [33]. The Hamming distance between two vectors A and B is calculated as SUM(XOR(A,B))–the sum of the element-wise XOR, the difference vector, of A and B (6 in Fig. 2). Such computations are exceedingly simple compared to the complex matrix multiplication-accumulation operations of NNs.

The vector primitives of HDC–multiply, add, permute (abbreviated as MAP)–enable the association of representations in the HD space via superposition while still allowing the individual component representations to be recovered [33], [36], [37]. The representational space is closed under the MAP primitives [38]. For binary hypervectors, the MAP primitives are implemented as element-wise XOR (multiply), element-wise thresholded accumulation (add), and the wrapped shifting of all elements of the vector $+1$ index (permute) [36]. For binary hypervectors, addition must be thresholded to return the resulting integer-valued vector elements to '1' or '0'. As the permute function is simply a shift in the indexing of vector elements, it is performed solely through wiring, requiring no extra hardware. This is demonstrated in the green portion of Fig. 3 for the letter hypervectors: here the letter 1 hypervector is permuted twice and the letter 2 hypervector is permuted once. See Section IV for details.

III. Machine Learning Benchmark Task

The SHDC architecture we present can easily be generalized to almost any ML, pattern recognition, or classification task by varying the length (and thus representational capacity) of the hypervectors, the number of classes stored in the item memory, and the projection function implemented by the encode module.

In the following sections, we focus on the European language recognition task [36], [56], [57] and tailor our architecture to its specifications. The input data is a text stream consisting of only the Latin alphabet and the space character, and the desired output is the language of the text. There are 21 European languages in the dataset resulting in $M = 21$ memory classes (one for each language).

As our dataset is comprised of 27 features known a priori, we implement our projection function as a lookup table mapping each feature of the data (character) to a randomized seed vector which serves as its representation in HD space. This lookup table that maps features of the dataset to hypervectors is termed the ***seed memory***. The details of how the hypervector representing an entire text stream is generated are discussed in detail below.

IV. Encode Module

We begin by describing the encode module (EM) of our SHDC circuit, shown on the left in Fig. 3. The EM implements the projection function mapping input data to the N-dimensional representational space. The letters of the input text are received one at a time and encoded into hypervectors representing groups of three consecutive letters–termed ***trigrams***–by the EM. Each letter as well as the space character is represented by a predefined N-dimensional ***seed vector*** stored in the ***seed memory*** of the EM. The seed memory is implemented as a look-up table; its architeture is shown in Fig. 4. All seed vectors are generated randomly and chosen to be approximately orthogonal to each other in the HD space in order to ensure all features of the dataset–characters in our application–have unique representations.

During the encode stage, input letters are first mapped to their corresponding seed vectors by the seed memory, then permuted according to their position in the trigram. The first letter hypervector is permuted twice, the second once, and the third not at all. This is to ensure that different combinations of the same letters have unique trigram hypervectors and thus representations within the HD space. As shown in Fig. 3, permutation is accomplished through the wiring between buffers as letter hypervectors propagate through them in FIFO order.

Next, the permuted letter hypervectors are bound into a single trigram hypervector using two subsequent vector

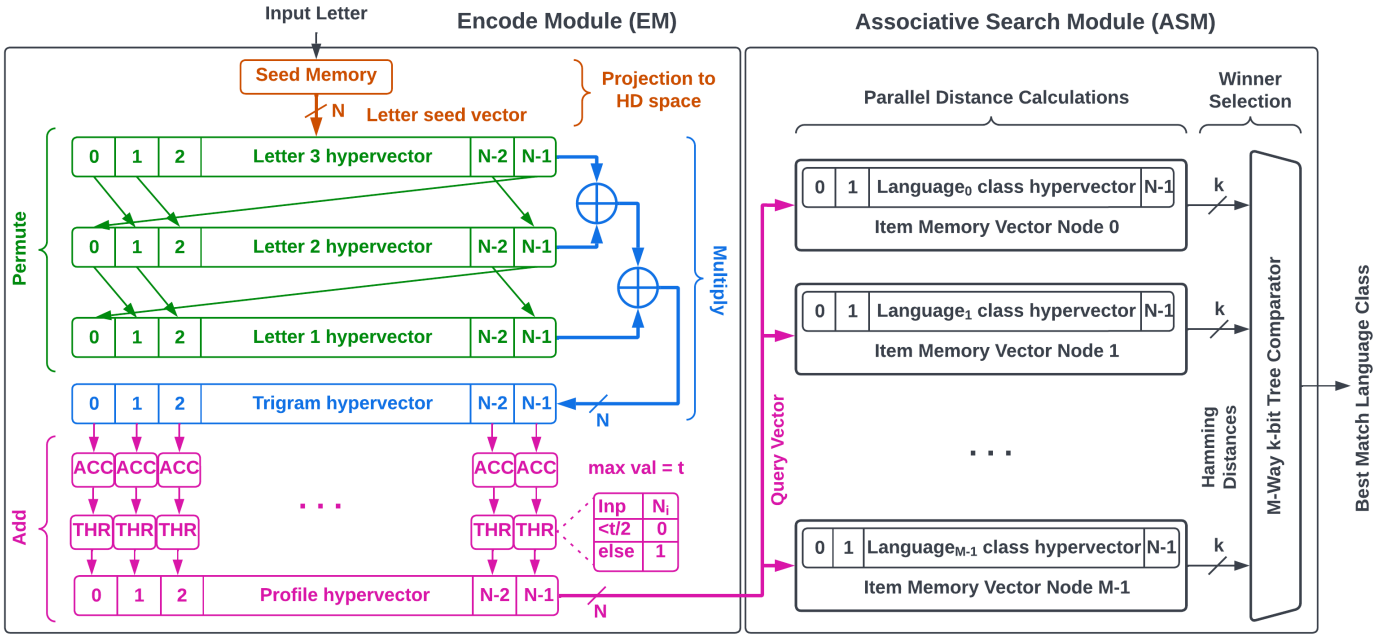

Fig. 3: HDC language recognition architecture.

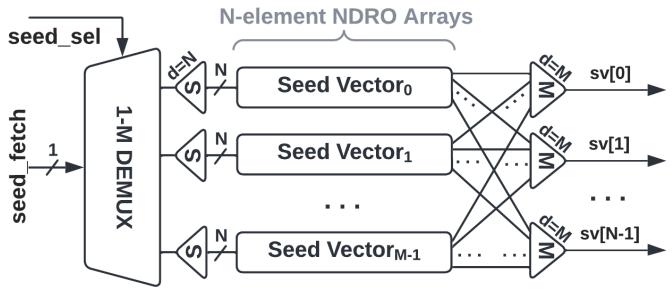

Fig. 4: Seed memory implementation. Each element of each seed vector is stored in its own non-distructive read out (NDRO) cell. All elements of the selected seed vector are read out in parallel (simultaneously) to their designated output wire; i.e. element i of the selected seed vector is driven to wire sv[i].

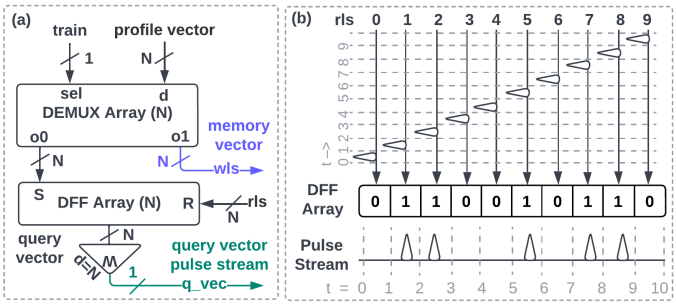

Fig. 5: (a) Encode module (EM) output routing. (b) Vector pulse stream formatting.

multiplication operations–implemented as XOR for binary vectors. Lastly, all trigram hypervectors of the text stream are accumulated element-wise into a single, integer hypervector such that element i if the integer hypervector is the sum of the i^{th} element of all binary trigram hypervectors. This is accomplished with a vector of N accumulators, one per element. A threshold of $\frac{t}{2}$ (where t is the total number of trigrams in the text) is then applied to each element of the integer profile vector in order to return it to the binary representational space by converting all elements smaller than $\frac{t}{2}$ to 0 and all others to 1. The resulting binary hypervector represents the entire text and is termed its ***profile vector***.

For scalability, we use inductor-based accumulator cells with temporal result readouts in conjunction with temporal thresholding gates each implemented with a single non-destructive read out (NDRO) cell. Accumulators produce their outputs x cycles after the start of the read operation where x is the number of pulses accumulated. With this temporal formatting, the threshold function is implemented as a temporal inhibit signal sent $\frac{t}{2}$ cycles after the start of the read operation to block output from any accumulators that have not yet produced it. A DFF array is used to temporally synchronize the elements of the profile vector following the thresholding operation.

How the EM outputs the profile vector depends on whether the language (class) of the profile vector is known or not. The implementation of profile vector routing at the output of the EM is shown in Fig. 5 (a). During training, when the language of the input text is known, its profile vector represents a learned class. In this case the profile vector–termed a ***memory vector***–will be sent to the writelines output of the EM (wls) were it will written to one of the M memory nodes (IMVNs) of the ASM, forming a new class. Each element of the hypervector is represented on its own wire for memory write operations, so the wls signal consists of N, 1-bit writelines. The ASM's sel control signals select the memory node to store the profile vector in, with $sel[i]$ indicating the write should be preformed on IMVN i. When the language of the input text is unknown, its profile vector–termed a ***query***

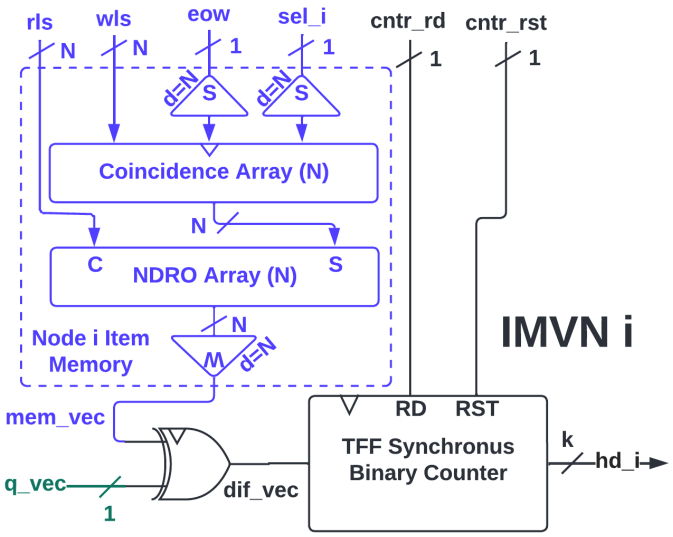

Fig. 6: Item memory vector node (IMVN) implementation. In-node memory used to store the memory vector for class i is shown in purple. Distance calculation logic is shown in black.

vector–is sent to the ASM for classification. In this case, the profile vector is directed to the query vector output (q_vec) for distribution to all M memory nodes of the ASM to perform the search.

Fig. 5 (b) shows how the N-bit query vector is formatted as a single pulse stream of length N. This is accomplished using N readlines (rls) to read the query vector out in a serial pattern such that the i^{th} element of the query vector is read in cycle i of the read operation. A merge tree with a fan-in degree of N then merges the N individual query vector element reads into a single, N-cycle pulse stream where the i^{th} element of the vector is represented in the i^{th} cycle of the pulse stream. Memory vector read operations within the IMVNs of the ASM are also performed as shown in Fig. 5 (b) to format memory vectors into pulse streams. Read operations in the IMVNs use the same readline control signals as the EM (with added path-balancing logic) to ensure the memory and query vector pulse streams are temporally synchronized within the IMVNs.

V. Associative Search Module (ASM)

In this section we present the associative search module (ASM) of our SHDC circuit. The ASM consists of M item memory vector nodes (IMVNs) and an M-argument comparator. Each IMVN stores a single class hypervector, calculates its Hamming distance from input query vectors, and outputs said distance as a k-bit binary number where $k = ceil(log_2(N))$. The comparator accepts the M, k-bit Hamming distances and selects the minimum, outputting the index of the corresponding class, indicating the best match to the query vector. The implementation and operation of the ASM are discussed below and shown on the right in Fig. 3. The implementation of the IMVNs of the ASM is shown in Fig. 6; the IMVN control signals shown are shared by all M IMVNs of the ASM.

A. Training

During memory write operations, the profile vector to be written to memory is driven to the ASM by the N writelines (wls) output by the EM. The writelines are projected to all M IMVNs, the M sel control signals of the ASM select which IMVN the profile vector will be written to.

Each IMVN has an N-element NDRO array to store its class vector. An N-element array of coincidence cells is used to gate inputs to the NDRO array to ensure the write operation is only performed to the IMVN indicated by the sel lines. Element i in the coincidence array of IMVN j accepts $wls[i]$ as its data input, $sel[j]$ as its select input, and eow as its reset input, used to signal the end of each write operation. The wls and eow signals are sent to all IMVNs for every write operation but only one of the M sel signals will fire during any given write so only that IMVN will have its memory written to.

B. Classification

1) Item Memory Vector Nodes (IMVNs): IMVNs (Fig. 6) implement the distance calculations of HDC. During search operations, the query vector is projected to all M IMVNs which perform their distance calculations in parallel. As mentioned previously, the query vector is output from the EM as a single, N-cycle pulse stream. The memory vectors stored in the IMVNs are read out in the same format using the same readline signals rls for temporal synchronization with the query vector read (Fig. 5(b)).

With query and memory vectors both represented as N-cycle pulse streams, their difference vector can be calculated using a single XOR gate (see Fig. 2 and 6). The difference vector pulse stream forms the input to the synchronous TFF counter. The counter accumulates all pulses of the difference vector–the Hamming distance between the query and memory vectors–over an accumulation period of N cycles (the temporal length of the difference vector pulse stream) and outputs this distance as a binary number. The maximum Hamming distance possible for vectors of length N is N, thus the bitwidth of the TFF counter is $k = ceil(log_2(N+1))$. A read signal ($cntr\_rd$) is used to gate output from the counter to ensure it only produces output once the accumulation is complete.

As the counter is synchronous, the output of each bit stage is offset by a single cycle. Therefore, the counter read signal ($cntr\_rd$) must follow the end of the accumulation period by k cycles to allow the LSB value to be fully updated before the read. The $cntr\_rd$ signal is shared by all IMVNs in order to synchronize their outputs.

2) Comparator: Each of the M IMVNs generates the Hamming distance between its stored memory vector and the query vector as a k-bit binary number. At the output of the ASM is a M-way k-bit tree comparator which selects the minimum of these Hamming distances and outputs the index of the corresponding IMVN–the index of the best matching class–as a binary number. To minimize design footprint, all k-bit comparators are implemented with a single 1-bit comparator which is used to compare arguments one bit at a time from MSB to LSB, stopping after an inequality is found or all bits have been compared.

VI. Methodology

We evaluate our SHDC architecture using a combination of WRspice, PyRTL, and analytical models. We use a cell library derived from the SUNY RSFQ cell library [4] with the exception of the NDRO cell which is from [58]. See Table V for a list of cells. We extracted the parameters for all gates in our library by simulating them at the circuit level in WRspice [59], an open-source SPICE simulator, using the open-source *MIT-LL SFQ5ee 10 kA/cm2* process [60]. We used our SPICE models to extract the JJ counts, critical currents, bias currents, propagation delays, hold times, and set-up times for all gates in our cell library.

In order to simulate and verify our design at scale, we built RTL models of our architecture as well as its individual modules in PyRTL [61], a python library for RTL design, simulation, tracing, and testing built on Verilog. We parameterize our RTL models with the gate parameters extracted from our circuit-level WRspice simulations in order to verify both the functional and timing correctness of our design as well as obtain latency, area, and power results for our full-scale models. We explicitly model all control, signal distribution, and clock tree hardware to ensure our designs are properly path-balanced, there are no timing violations, and our results are accurate.

Latency results for our design are obtained directly from the parameterized, ps-scale RTL simulations of our designs. We extract gate counts from our RTL models and use these in tandem with the gate parameters extracted from our SPICE models to obtain area–in terms of JJ counts–and power consumption results for our designs.

To obtain mm^2 area estimates for our SHDC architecture and its component modules under the *MIT-LL SFQ5ee 10 kA/cm2* manufacturing process, we use the gate count data of our designs in combination with MIT Lincoln Lab (MIT-LL) JJ area data which gives the physical area of JJs based on their critical currents. We calculate the μm^2 area of all gates in our cell library from the JJ area data and the critical current of each JJ in each gate taken from their circuit-level implementations. We obtain the physical area of our designs using this gate area data in tandem with our module gate count data. We also introduce a $3\times$ area overhead to estimate the effect of non-JJ components as well as routing and layout restrictions.

To obtain power consumption results, we wrote a python library to calculate the dynamic and static power consumption of each cell in our library given the critical currents of its JJs and bias resistances, as well as the bias voltage and operating frequency of the chip. All modules in our design use the same bias voltage of 10 mV. Table V shows the per-gate static and dynamic power consumption values for all cells in our library at an operating frequency of 33.3 GHz and the gate-wise power consumption breakdown of our full SHDC circuit for $N = 1k$.

It is worth noting that static power dissipation dominates the total power consumption of RSFQ circuits due to the large amount of power dissipated by the bias resistors [62], [63]. The static power dissipation of the cells in our library is $\sim 10^2$ times higher the dynamic power dissipation at the chip's operating frequency of 33.3 GHz (see Table V). In recent years, energy-efficient rapid single flux quantum (ERSFQ) circuits have emerged as a highly promising alternative to traditional RSFQ circuits. In this technology, bias resistors are replaced with bias JJs, completely eliminating static power dissipation [62], [63]. The timing characteristics and physical area of ERSFQ gates are assumed to be the same as those of their RSFQ counterparts as their gate structures are the same, just the bias current supply lines differ [15], [62].

Following the methodology in [15], to calculate the power consumption of our design's implementation in ERSFQ technology, we estimate that ERSFQ gates dissipate twice the dynamic power of their RSFQ counterparts and zero static power. ERSFQ power consumption results for our SHDC design are shown in Table VI for the language recognition benchmark ($M = 21$) and Table VIII for the ImageNet benchmark ($M = 1k$). All other results are for the RSFQ implementation of our design.

Without exception, all power calculations given in this paper for our SHDC design assume a worst-case activity factor, meaning each JJ within every gate of our circuit is assumed to be accessed every clock cycle. This is a large over-assumption, however, it effects only the dynamic power consumption. Therefore, the impact to RSFQ total power consumption is minimal but the impact to ERSFQ total power consumption values is substantial since all power dissipation is dynamic in this technology.

For both RSFQ and ERSFQ implementations, we calculate cooling power consumption as 395 W of cooling power per W of chip power for a helium reliquifier refrigeration system [64].

VII. Evaluation

We assess the performance of our SHDC architecture for vector lengths in the range $N = 1 - 10k$ as this spans a typical range of vector lengths used in most HDC applications [33], [38], [40]. As mentioned above, all power numbers reported for our SHDC circuit assume a maximum activity factor, meaning every JJ is assumed to activate every clock cycle. Additionally, control, signal amplification, and clock tree hardware are accounted for in all results. All SHDC results are for the European language recognition task ($M = 21$) unless otherwise indicated. Power results are for the RSFQ implementation of our circuit and include cooling power unless otherwise indicated. Area percentages are based on JJ count areas as described in the previous section.

The maximum operating frequency of the SHDC associative memory chip is 33.3 GHz, set by the IMVN. The only module that operates at a different frequency is the comparator at the output of the ASM which uses a self-clocking scheme and operates at 6.67 GHz with worst-case latency. See Section VII-B below for details. The area and power consumption of our SHDC design broken down to its individual modules as well as memory versus logic are shown in Fig. 7 and Fig. 8 respectfully. Individual gate contributions to area and power consumption totals for $N = 1k$ are shown in Table V. The area scaling results for SHDC as well as its component modules in terms of both JJ count and mm^2 are shown in Table IV. Performance and power scaling of our design

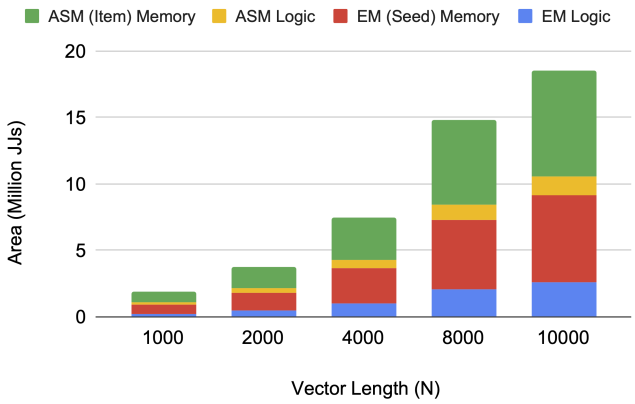

Fig. 7: Superconducting HDC (SHDC) area scaling.

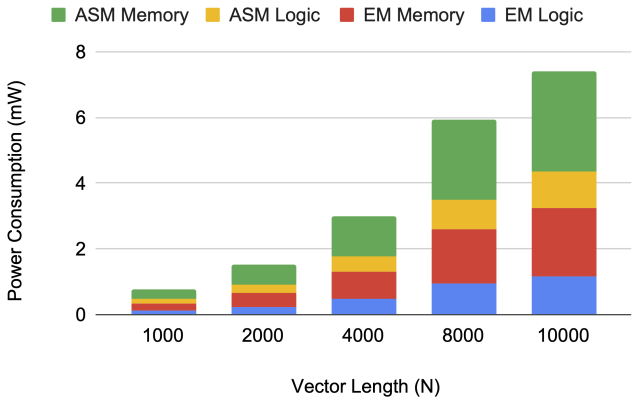

Fig. 8: SHDC power scaling. Includes cooling power.

are shown in Tables II and III respectively. ERSFQ power consumption results for our SHDC circuit are given in the CMOS and SuperNPU comparison sections and shown in Tables VI and VIII.

A. Encode Module

Two factors affect the area of the EM, predominantly the length of hypervectors (N) and to a lesser extent, the length of the input text. As the vast majority of the EM is comprised of gate arrays of length N, its area scales approximately linearly with N. The length of the text being encoded in trigrams (groups of three consecutive characters), t, also has a minor effect on EM area as the accumulators must be adequately sized to hold up to t pulses–one per trigram hypervector.

Sized for 1kB training texts, the EM comprises 47-49% of the area of the full design over a range of $N = 1 - 10k$. The seed memory (Fig. 4) comprises 72% of the total EM area over this range. The power consumption of the EM is proportional to its area. Operating at 33.3 GHz, the EM accounts for 42-44% of the power consumption of the full SHDC design over a range of $N = 1 - 10k$ with the seed memory accounting for 64% of the EM's total power consumption. See Fig. 7 and Fig. 8.

Due to the fact that all bits of hypervectors are operated on in parallel in the encode operation implemented by the EM (see Fig. 3), the EM's cycle-based latency is determined solely by the number of trigams in the input text, t. Each trigram hypervector takes four cycles to encode, resulting in $4t$ cycles to accumulate all trigram hypervectors. The temporal accumulation and thresholding operations–performed simultaneously–take $\frac{t}{2}$ cycles. Thus, the total latency of the EM–the projection of a t-trigram input text to HD space–takes $4.5t$ cycles.

B. Associative Search Module (ASM)

The ASM consists of the M IMVNs and the comparator used to select the minimum distance class. The ASM comprises 51-53% of the design footprint and 56-58% of the power consumption for vector lengths ranging from $N = 1{-}10k$. The item memory of the ASM, consisting of all in-node memories (Fig. 6) of the M IMVNs, comprises 79-85% of the ASM's total area and 68-73% of its total power consumption over this range. The total area of the IMVNs comprises 83-85% of the ASM footprint and the comparator comprises 3.7-0.5% over the same range of vector lengths. The remainder of the area is signal amplification and control circuitry hardware. As all IMVNs perform their distance calculations in parallel, the total latency of the ASM is the sum of the IMVN and comparator latencies, discussed in detail below.

1) Item Memory Vector Nodes (IMVNs): IMVN area scales approximately linearly with the vector length, N. The area of the TFF counter scales proportionally to $log_2(N)$; however, for large values of N, IMVN area becomes dominated by the in-node memory (Fig. 6) which scales proportionally to N. The in-node memory comprises 95-99% of the area and 94-99% of the power consumption of the IMVNs over the range of $N = 1 - 10k$. The total area of the IMVNs dominates the area of the ASM at all scales of N.

As discussed in Section V-B, with both the query and memory vectors formatted into N-cycle pulse streams, it takes N cycles to compute their difference vector using a single XOR gate. The difference vector pulse stream is accumulated by the TFF counter as it is generated. Due to the synchronous nature of the TFF counter, its read must occur k cycles after the completion of the accumulation where $k = log_2(N)$ is the bitwidth of the counter. Thus the total latency of the IMVNs is $N + k$ cycles. The IMVN distance calculation latency comprises 78-96% of the total ASM classification latency over vector lengths in the range $N = 1 - 10k$.

2) Comparator: The area of the binary tree comparator discussed in Section V-B scales approximately linearly with the number of arguments M and to a lesser extent logarithmically with the argument bit width, $k = log_2(N)$. Because k-bit comparators are implemented using 1-bit comparators, the latency of the binary comparator is logarithmically proportional to both the number of arguments, M, and the length of the vectors, N. The latency of the k-bit comparators depends entirely on the bit at which the arguments differ, the worst case being equal arguments which take $k + 1$ cycles to compare.

In order to minimize latency, we use a self-clocking scheme for both the k-bit comparator and the M-way tree comparator

TABLE II: **SHDC performance scaling.** The minimum throughput between the EM and ASM determining overall design throughput is shown in bold.

$Vector Length (N)$	1000	2000	4000	8000	10000
EM Throughput (M enc ops/s)	**22.24**	22.24	22.24	22.24	22.24
ASM Throughput (M searches/s)	25.94	**14.42**	**7.69**	**3.99**	**3.21**
SHDC Throughput (MCO/s) (overall)	22.24	14.42	7.69	3.99	3.21

TABLE III: SHDC power consumption (μW).

$Vector Length (N)$	1000	2000	4000	8000	10000
Dyn. Power	0.026	0.052	0.102	0.204	0.255
Static Power	1.92	3.76	7.42	14.75	18.42
Total Power (w/o cooling)	1.95	3.81	7.53	14.95	18.67
Cooling Power	7.68E+2	1.50E+3	2.97E+3	5.91E+3	7.38E+3
Total Power (w/ cooling)	7.70E+2	1.51E+3	2.98E+3	5.92E+3	7.39E+3

in which the result of the slowest comparison at each stage is used to initiate the comparison(s) in the next stage. This self-clocking scheme enables us to capitalize on the latency variability of the comparison operations by triggering the next clock cycle as soon as the current cycle's computation is complete. The maximum length of a single comparison cycle is 150 ps corresponding to a minimum operating frequency of 6.67 GHz. With worst-case latency–maximum number of comparison cycles at maximum cycle length–the 21-way k-bit comparator comprises 3.6-21% of the total latency of the ASM over vector lengths in the range $N = 1 - 10k$.

As the comparator is used to select the winning class at the output of the ASM, it is the last module in our SHDC circuit meaning its self-clocking scheme does not cause synchronization issues with the rest of the circuit. We assume worst-case latency for the comparator–6.67 GHz operating frequency–for all results given in this paper.

C. Full SHDC Design

1) Memory Footprint: In HDC designs, only the S seed vectors of the seed memory (Fig. 4) and the M class vectors of the item memory (Fig. 6) must be stored. With binary vectors of length N, this totals to $N*(S+M)$ bits. For the given task, we have $S = 27$ and $M = 21$ for a total memory footprint of only $48N$ bits. In both the seed and item memories, each bit of memory is implemented with a single NDRO gate. Over the range $N = 1 - 10k$, the memory NDROs comprise 37% of the item memory and 58% of the seed memory with the rest of the area being comprised of read and write control circuitry. Despite the small memory requirements of the SHDC architecture, the combined seed and item memory footprints still comprise 75-78% of the full design footprint due to the simplicity of the computational logic of HDC.

2) Performance: Performance is expressed in terms of millions of classification operations per second (MCO/s). Results are shown in Table II. Each classification consists of an encode operation–performed by the EM–followed by a search operation–performed by the ASM. These two stages can be overlapped, thus the overall throughput of the circuit is set by the throughput of the slowest stage.

The latency of the EM is proportional to the length of the input text being encoded. With 1kB training texts, operating at 33.3 GHz, the EM can perform 22.24 million (M) encode operations per second for all values of N. The latency of the ASM is proportional to N so its throughput decreases as vector length increases. The throughput of the ASM becomes the limiting factor on overall throughput for $N > 1k$. See Table II.

TABLE IV: **SHDC area scaling.** The upper bounds of single-chip fabrication limits with both on-chip memory (total area) and off-chip memory (logic area only) for each module are shown in bold text. See Section VII-D for details.

$Vector Length (N)$	1000	2000	4000	8000	10000
IMVN Logic (JJs)	2.1k	2.36k	2.67k	3.01k	3.36k
IMVN Mem (JJs)	38k	76k	152k	304k	380k
IMVN Total (JJs)	40k	78k	155k	307k	383k
IMVN Logic (mm^2)	0.12	0.13	0.15	0.17	0.19
IMVN Mem (mm^2)	2.21	4.43	8.86	17.71	22.14
IMVN Total (mm^2)	2.33	4.56	9.01	17.88	22.33
EM Logic (JJs)	0.26M	0.52M	**1.03M**	2.06M	2.58M
EM Mem (JJs)	0.65M	1.31M	2.61M	5.23M	6.53M
EM Total (JJs)	**0.91M**	1.82M	3.65M	7.29M	9.11M
EM Logic (mm^2)	16.3	32.6	**65.2**	130	163
EM Mem (mm^2)	32.1	64.1	128	256	320
EM Total (mm^2)	**48.4**	96.8	193	387	484
ASM Logic (JJs)	0.21M	0.36M	0.63M	**1.18M**	1.45M
ASM Mem (JJs)	0.80M	1.60M	3.19M	6.38M	7.98M
ASM Total (JJs)	**1.01M**	1.95M	3.82M	7.56M	9.43M
ASM Logic (mm^2)	13.7	23.4	42.2	**79.4**	98.2
ASM Mem (mm^2)	46.5	93.0	186	372	465
ASM Total (mm^2)	**60.3**	116	228	451	563
SHDC Logic (JJs)	0.47M	**0.87M**	1.66M	3.24M	4.03M
SHDC Mem (JJs)	1.45M	2.90M	5.81M	11.61M	14.51M
SHDC Total (JJs)	1.92M	3.78M	7.47M	14.85M	18.54M
SHDC Logic (mm^2)	30.0	**56.0**	107	210	261
SHDC Mem (mm^2)	78.6	157	314	628	786
SHDC Total (mm^2)	107	213	422	838	1047

D. SHDC Fabrication Bounds

As discussed in Section I, current RSFQ fabrication limits depend on the complexity of the circuit in question, lying at ∼20-30k JJs for logically complex designs [9] and over 1M JJs for highly-regular designs [6]–[8]. Although much of the SHDC design consists of large logic gate and memory arrays with highly regular structures, its control circuitry and the comparator module are fairly complex. As such, one would expect the actual fabrication limits for SHDC to lie close to but likely below 1M JJs. Thus, we use ∼1M JJs as the upper bounds for the design scales at which SHDC and its component modules are realistically fabricatable.

The upper bounds (∼1M JJs) of design scales for which SHDC and its component modules are realistically fabricatable with both on-chip memory (total area) and off-chip memory (logic area only) are shown in bold in Table IV. HDC applications are practically useful at vector lengths of $N \geq 500$ [33],

Gate	JJs	Dyn. Power 33.3GHz (μW)	Static Power (μW)	EM Gates	IMVN Gates	Comparator Gates*	Gate Totals	JJ Totals	Chip Dyn. Power** (μW)	Chip Static Power (μW)	Total Power (μW)
JTL	2	3.45E-8	3.45E-6	6090	5917	1335	13342	26684	4.23E-4	4.60E-2	4.64E-2
split	3	5.89E-8	5.99E-6	52121	87196	3227	142544	427632	8.25E-3	8.54E-1	8.62E-1
merge	7	8.20E-8	5.00E-6	26999	21399	1222	49620	347340	3.99E-3	2.48E-1	2.52E-1
inhibit	8	1.43E-7	7.77E-6	1000	0	0	1000	8000	1.43E-4	7.77E-3	7.91E-3
Inverter	10	1.43E-7	7.77E-6	3028	0	60	3088	30880	4.36E-4	2.40E-2	2.44E-2
Coincidence	11	1.97E-7	1.15E-5	6056	21000	440	27496	302456	5.35E-3	3.15E-1	3.20E-1
XOR	9	1.21E-7	1.74E-6	2000	21	20	2041	18369	2.46E-4	3.55E-3	3.80E-3
LA	6	1.12E-7	8.83E-6	0	0	8	8	48	1.79E-7	7.07E-5	7.09E-5
DFF	6	6.72E-8	3.68E-6	2000	2079	93	4172	25032	2.75E-4	1.53E-2	1.56E-2
TFF2	10	6.49E-8	1.48E-5	0	210	0	210	2100	1.36E-5	3.10E-3	3.11E-3
NDRO	14	1.36E-7	7.62E-6	27000	21840	760	49600	694400	6.67E-3	3.78E-1	3.84E-1
uACC***	42	2.93E-7	2.49E-5	1000	0	0	1000	42000	2.93E-4	2.49E-2	2.52E-2
Totals								**1924941**	**2.61E-2**	**1.92**	**1.95**

TABLE V: **SHDC gate-wise power consumption breakdown (M=21, N=1k).** The circuit runs at 33.3 GHz with the exception of the comparator (*) which runs at 6.67 GHz at minimum. Thus, the dynamic power consumption numbers for the full chip (**) are slightly lower than they would be if all gates were clocked at 33.3 GHz. The *IMVN Gates* column gives the gates for all M IMVNs including their control circuitry, clock trees, and signal distribution hardware. The *Comparator Gates* column also includes the gates for the comparator's control circuitry, clock trees, and signal distribution hardware. *Comparator Gates* are assumed to operate at the minimum frequency of 6.67 GHz here. The total power numbers in the last column do not include cooling cost. ***The unipolar accumulator gate (uACC) consists of a DFF, 2 NDROs, 2 splits, and 2 additional JJs.

TABLE VI: CMOS power consumption comparison (μW).

Vector Length(N)	1000	2000	4000	8000	10000
CMOS HDC	1.1E+5	2.4E+5	4.9E+5	9.5E+5	1.09E+6
RSFQ SHDC (w/o cooling)	1.95	3.81	7.53	15.0	18.7
RSFQ SHDC (w/ cooling)	7.7E+2	1.51E+3	2.98E+3	5.92E+3	7.39E+3
ERSFQ SHDC (w/o cooling)	0.052	0.103	0.205	0.408	0.510
ERSFQ SHDC (w/ cooling)	20.7	40.8	81.1	162	202

[38], so as long as the vector length (N) for which a module is realistic to fabricate lies at or above $N = 500$, we consider the module fabricatable at practically useful scales. The EM and ASM with both on- and off-chip memory, as well as the full SHDC design with off-chip memory are all realistic to fabricate at practically useful scales of $N \geq 1000$. It is possible to fabricate individual IMVNs at practically useful scales even using the lower bounds of 20-30k JJs. Although it is not shown in the table, the full SHDC architecture with on-chip memory is also realistic to fabricate at a practically useful scale of $N = 500$ having 995k JJs, although it lies close to the upper bound.

To implement SHDC at scales beyond $N = 500$, one could fabricate the component modules of SHDC on separate chips to create multi-chip modules connected by transmission lines with SFQ pulses. With the EM and ASM each fabricated on their own chip and connected in this manner, SHDC could be implemented for scales of up to $N = 1000$ with on-chip memory and $N = 4000$ with off-chip memory.

E. CMOS Comparison

To facilitate a comparison of our SHDC design with an equivalent CMOS circuit, we benchmark our circuit against the 2D CMOS HDC circuit architecture presented in [36] for the European language recognition task [56], [57]. Recall that there are 21 classes in this task ($M = 21$). This design implements the same algorithm as our SHDC design using the *TSMC 65nm LP CMOS* process.

1) Performance: The study of [36] for a CMOS HDC design focuses on the ASM module and in particular the IMVNs. As in our SHDC IMVNs, the CMOS IMVNs presented in [36] use a single XOR gate to compare one element of the query and memory vectors at a time, requiring O(N) cycles to compute the Hamming distance between the two hypervectors. In order to compare against [36], we assume zero latency for the circuitry that is not described in the CMOS design, namely the EM and combinational comparator used in the ASM. We assume that the CMOS distance calculation takes exactly N cycles. Thus, the latency of the CMOS design given here is optimistic even for the distance calculation alone. Our SHDC design's latency includes all components. As such, this is a highly favorable assumption for the CMOS design.

Since performance metrics are not reported in [36], we assume the CMOS design runs at an aggressive operating frequency of 5 GHz and that the entire classification requires N cycles. Even under these conditions, our SHDC design outperforms the CMOS HDC design by 78-84% for vector lengths in the range of $N = 1 - 10k$. Even assuming the CMOS HDC design can operate at a frequency of 10 GHz, our design still outperforms it by 55-69% over this range.

Given the aggressiveness of our latency-based assumptions in favor of the CMOS design, the gains given above are entirely due to the faster operating frequency of SHDC. If we were taking into account the latency introduced by the CMOS comparator, which accounts for 3.6-21% of the SHDC ASM's latency, we would also expect to see cycle-based gains from the self-clocking scheme utilized by the SHDC comparator.

TABLE VII: Memory footprint and performance of SuperNPU versus SHDC for ImageNet.

Benchmark	M Params	Mem (Mb)	M MACs	M Classific./s
AlexNet	61	488	720	1.17
GoogLeNet	6.8	54.4	1550	0.54
MobileNet	1.32	10.56	76	11.08
ResNet 50	25	200	3860	0.22
VGG16	138	1104	15300	0.06
SHDC N=1k	1	1	-	21.37
SHDC N=10k	10	10	-	3.10

2) Power Consumption: Table VI shows how the power consumption of both RSFQ and ERSFQ implementations of our SHDC design with and without cooling cost compare to that of the CMOS HDC benchmark design over the range $N = 1 - 10k$. Without cooling cost, RSFQ SHDC consumes four orders of magnitude less power than CMOS HDC. Including cooling cost, this factor drops to two orders of magnitude. ERSFQ SHDC consumes six orders of magnitude less power than CMOS HDC without cooling cost and three orders orders of magnitude less power including cooling.

F. SuperNPU Comparison

In this section we evaluate SHDC's performance as a superconducting ML technique against that of the state of the art SNNA: SuperNPU. The NNs used to benchmark SuperNPU were designed for use with the ImageNet dataset [65] which contains images belonging to a total of 1000 classes. Tailoring our SHDC design to this benchmark, the ASM contains $M = 1000$ IMVNs representing the 1k classes and a 1k-way comparator to select the winning class.

As the input data consists of images for the ImageNet task, it makes more sense to use a pixel-value based projection function to perform the mapping of images to the HD space than a look-up table mapping data features to predefined seed vectors as we do not know the features of the dataset a priori.

Under this framework, during training input images of a given class would be projected to hypervectors one at a time based on their pixel values and then accumulated into a single profile hypervector representing the class just as with the trigram hypervectors for the language recognition task in what is termed an N-gram encoding [49]. However, for classification, the input is a single image so no accumulation stage is required; the image is mapped to the HD space and sent directly to the ASM for classification.

The area of the EM scales only with hypervector length (N) and the number of object hypervectors in a class (t) but not with the number of classes (M). Therefore, we would expect an EM implementing a simple pixel-based projection function to be roughly the same scale of complexity as the EM for the language recognition task presented above, given that it does not include a seed memory which comprises $\sim 37\%$ of the number of JJs of the language recognition EM. To be conservative, we estimate that the EM for the ImageNet task has the same area (in terms of JJ count) and power consumption of the $M = 1k$ ImageNet ASM. Recall that the EM is smaller than the ASM for the language recognition task for which $M = 27$. As $M = 1k$ for the ImageNet task and the area of the EM does not scale with M while that of the ASM does, this is likely a significant over-estimate.

1) Memory Requirements: The number of parameters for five popular NN architectures used to benchmark SuperNPU are shown in Table VII. All parameters are quantized to 8 bits in SuperNPU; the resulting memory requirements of each network under this quantization scheme are shown in the same table. Network parameters are stored in off-chip DRAM in the SuperNPU architecture. Additionally, SuperNPU uses two buffers of 24 MB each to store partial calculation results and an additional buffer of 128 kB to hold the network parameters currently in use for a total of 48.128 MB of on-chip memory (see Table I in [15]).

With $M = 1000$ classes and vectors of length N, our SHDC architecture requires $N \times M = 1000N$ bits of memory to store the 1k binary class vectors in the ASM. As the EM does not contain a seed memory for this application, the item memory of the ASM comprises the memory footprint of the entire design. For vectors in the range $N = 1 - 10k$, the total memory requirements of SHDC's architecture for ImageNet would be $1 - 10$ Mb. Even at the extreme of $N = 10k$, the memory requirements of SHDC are still smaller than those of even the smallest NN architecture, MobileNet (see Table VII).

2) Area: The mm^2 area results given for SuperNPU in [15] ($\sim 299mm^2$) assume the JJ device technology used to implement SuperNPU is equivalently scaled to a 28 nm CMOS technology, which was used to benchmark the TPU [15]. To avoid making device technology assumptions for our design, we compare area against SuperNPU in terms of JJ counts.

As no JJ count numbers are given in the SuperNPU paper [15], we estimate them here given the mircroarchitectural details presented in their paper in combination with their cited multiplier and adder implementations [9], [66]. SuperNPU consists of a 64×256 processing element (PE) array, 48.128 MB of on-chip memory implemented with shift registers, and a data alignment unit (see Figures 3 & 19 and Table I in [15]). The PEs each contain a 20.3k JJ 8-bit pipelined multiplier [9], a 3k JJ adder [66], and 8 shift registers. Ignoring the shift registers, control circuitry, and clock distribution hardware, each PE uses ∼23k JJs. Thus, there are 64×256×23k JJs = ∼377M JJs total in the PE array. Again ignoring the control circuitry and clock distribution hardware, we will assume the shift-register based on-chip memory requires only a single JJ per bit. Under these assumptions, we have 48.128 MB×8 b/B = ∼385M JJs in the on-chip memory. We ignore the data alignment unit giving an optimistic total of 377M + 385M = 762M JJs for the SuperNPU architecture.

With $M = 1k$ for the ImageNet task, our SHDC design uses 96M JJs at small design scales ($N = 1k$) and 892M JJs at large scales ($N = 10k$). SHDC has a similar footprint to SuperNPU at large design scales, however, it makes use of entirely on-chip memory while SuperNPU requires additional off-chip DRAM.

As the area of SHDC scales proportionally to both the length of hypervectors used (N) and the number of classes (M), its area does not scale as favorably as that of SNNAs when the number of classes becomes very large. SNNAs have a fixed

TABLE VIII: Power consumption (W) of SuperNPU versus SHDC for ImageNet.

Architecture	SuperNPU	SHDC N=1k	SHDC N=10k	SHDC 1k Factor Improvement	SHDC 10k Factor Improvement
RSFQ (w/o cooling)	964	1.06E-04	9.87E-04	9.08E+06	9.77E+05
RSFQ (w/ cooling)	3.82E+05	4.20E-02	3.91E-01	9.08E+06	9.77E+05
ERSFQ (w/o cooling)	1.9	2.74E-06	2.62E-05	6.94E+05	7.24E+04
ERSFQ (w/ cooling)	751	1.08E-03	1.04E-02	6.93E+05	7.22E+04

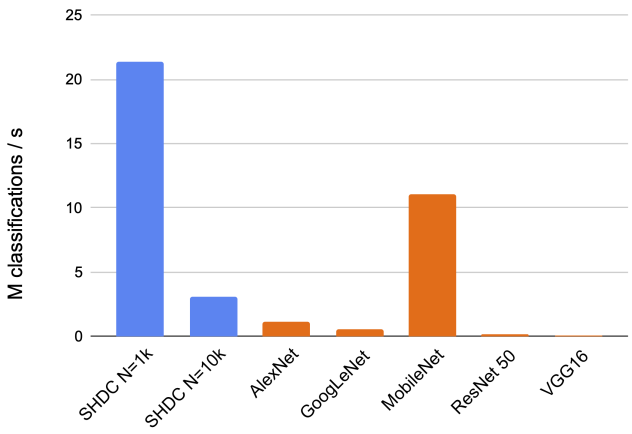

Fig. 9: SHDC SuperNPU performance comparison. The performance of our SHDC architecture designed for use with the ImageNet dataset at small ($N = 1k$) and large ($N = 10k$) design scales is shown in blue. SuperNPU's performance on five NN architectures is shown in orange assuming the peak performance of 842 TMAC/s is maintained throughout the computation.

size and instead suffer in terms of latency when accelerating very large NN models.

3) Performance: SuperNPU is reported to operate at 52.6 GHz [15]. Its performance in terms of millions of classifications per second (MC/s) for five popular NN architectures is shown in Table VII. The performance numbers given here assume that the SuperNPU architecture performs all computations at its peak performance of 842 TMAC/s. However, the actual performance of the SuperNPU architecture depends on the percent of the PE array being utilized for each MAC operation which depends on the architecture of the network being accelerated [15]. SuperNPU cannot maintain peak performance for the entirety of the computations for any of the NNs shown here so these performance numbers are optimistic in favor of SuperNPU.

For classification, input data will consist of only one image at a time, meaning the encode operation will be a single image projection operation without the accumulation and thresholding of multiple images into a single hypervector that is required during training. Thus, the latency of the EM will be minimal compared to that of the ASM even at small vector lengths, meaning the performance of the design will be set by the ASM. Assuming maximum latency for the comparator, the ASM can perform 21.37-3.1 MC/s over vector lengths ranging from $N = 1 - 10k$. Fig. 9 shows how the performance of our SHDC architecture compares to that of SuperNPU running the five NN architectures from Table VII at its peak performance of 842 TMAC/s.

4) Power Consumption: The power consumption results for both RSFQ and ERSFQ implementations of the SuperNPU and SHDC architectures with and without cooling power are shown in Table VIII. SuperNPU power consumption results come directly from Table III of [15]. At small scales ($N = 1k$) the RSFQ implementation of SHDC is $\sim 10^7\times$ more energy efficient than SuperNPU while the ERSFQ implementation is $\sim 10^6\times$ more energy efficient. At large scales ($N = 10k$) the RSFQ implementation of SHDC is $\sim 10^6\times$ more energy efficient than SuperNPU while the ERSFQ implementation is $\sim 10^5\times$ more energy efficient.

To summarize, SHDC tailored to the ImageNet ML task with 1k classes is $\sim 10^7 - 10^6\times$ more energy efficient than SuperNPU given a RSFQ implementation and $\sim 10^6 - 10^5\times$ more energy efficient than SuperNPU given an ERSFQ implementation for vector lengths in the range $N = 1 - 10k$. ***With very large vectors of $N = 10k$, our SHDC architecture outperforms SuperNPU in terms of throughput for all but the smallest NN architecture, MobileNet. With smaller vectors $N = 1k$, SHDC outperforms SuperNPU for all benchmarks.***

VIII. Conclusion

Given the extreme computational complexity, memory requirements, and training costs of NNs, such ML approaches are not ideal for implementation in superconducting digital technologies due to the area limitations arising from limited device density and the lack of scalable cryogenic memory solutions. Additionally, superconducting NN accelerators (SNNAs) are only designed to implement the forward pass of NNs and must be initialized with pretrained, quantized NN parameters, *making them fundamentally incapable of online learning*.

In contrast, hyperdimensional computing (HDC) uses simple computations and drastically less memory than SNNAs. Despite the small memory footprint, the computational logic of HDC is so simple that the design footprint is still dominated by memory. Furthermore, training and classification are the same under HDC, the only difference being in whether a vector is written to memory or searched for across the vectors already stored in memory. As such, training is exceedingly simple and can easily be performed online. This allows SHDC to continue learning after deployment in cryogenic environments, greatly increasing its adaptability and robustness.

Here we present the first superconducting HDC (SHDC) design and evaluate it against an analogous CMOS design as well as the state of the art SNNA, SuperNPU. As we have shown, the proposed SHDC circuit uses entirely on-chip RSFQ memory which is tightly integrated with logic,

operates at 33.3 GHz, is applicable to general ML tasks, and is manufacturable at practically useful scales given current SFQ fabrication limits. SHDC is 10^2 (RSFQ) to $> 10^3$ (ERSFQ) times more energy efficient than an analogous CMOS HDC circuit including cooling, $> 10^4$ (RSFQ) to 10^6 (ERSFQ) times more energy efficient than CMOS excluding cooling, and up to 10^7 (RSFQ) to 10^6 (ERSFQ) times more energy efficient than the state of the art RSFQ SNNA, SuperNPU, while achieving 78-84% higher throughput than the CMOS benchmark and 62-99% higher throughput for all but the smallest NN accelerated by SuperNPU.

To the best of the author's knowledge, SHDC is the only superconducting ML approach that is currently feasible at practically useful scales for real-world ML tasks and capable of online learning. Given the much greater maturity of semiconductor digital computing technologies over superconducting ones as well as the restrictions that come with cryogenic computing environments–predominantly the cost of cooling–SHDC is certainly not the best approach for all ML applications; however, we argue that it is a superior option for implementing ML in superconducting digital computing.

SHDC stands poised to make significant impact in any superconducting environment in which ML is required, especially given the fact that these systems are already cryogenic, meaning there is no cooling cost overhead in such environments. The on-chip SFQ memory, extreme noise and error tolerance, and high throughput performance of SHDC are particularly desirable for implementing ML solutions in these environments. Quantum control and error correction are two such key areas of active research.

IX. Future Work

The binary magnitude comparator implementation used in our SHDC design is rather basic, contributing a significant amount of area and latency to our design. This is particularly true at small design scales where the distance calculation latency is low and signal amplification and clock distribution hardware comprise a smaller proportion of the design. Using a more intelligently designed SFQ comparator could significantly decrease the area and latency of SHDC, especially at small design scales.

Under temporal logic schemes such as Race Logic [67], values are encoded in the temporal domain and MIN and MAX are first-order functions with highly simple implementations. This makes the implementation of distance-based computations that are at the heart of HDC extremely efficient. Our SHDC architecture stands to benefit greatly from the use of such a temporal logic scheme for the implementation of the distance calculations and comparisons of HDC.

X. Acknowledgements

This work was supported by the Army Research Office (ARO) under Proposal No. FP00014632, "CS Accelerators ARO #3 with DOD NATIONAL SECURITY AGENCY (NSA)" as well as the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE- AC02-05CH11231.

References

[1] S. S. Tannu, P. Das, M. L. Lewis, R. Krick, D. M. Carmean, and M. K. Qureshi, "A case for superconducting accelerators," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, ser. CF '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 67–75. [Online]. Available: https://doi.org/10.1145/3310273.3321561

[2] V. Michal, E. Baggetta, M. Aurino, S. Bouat, and J.-C. Villegier, "Superconducting RSFQ logic: Towards 100GHz digital electronics," in *Proceedings of 21st International Conference Radioelektronika*, 2011.

[3] M. Pedram, "Superconductive single flux quantum logic devices and circuits: Status, challenges, and opportunities," in *2020 IEEE International Electron Devices Meeting (IEDM)*, 2020, pp. 25.7.1–25.7.4.

[4] "Suny rsfq cell library," accessed: 2023-03-28. [Online]. Available: http://www.physics.sunysb.edu/Physics/RSFQ/Lib/contents.html

[5] S. K. Tolpygo, "Superconductor digital electronics: Scalability and energy efficiency issues (review article)," *Low Temperature Physics*, vol. 42, no. 5, pp. 361–379, may 2016. [Online]. Available: https://doi.org/10.1063%2F1.4948618

[6] V. K. Semenov, Y. A. Polyakov, and S. K. Tolpygo, "New ac-powered sfq digital circuits," *IEEE Transactions on Applied Superconductivity*, vol. 25, no. 3, pp. 1–7, 2015.

[7] ——, "Ac-biased shift registers as fabrication process benchmark circuits and flux trapping diagnostic tool," *IEEE Transactions on Applied Superconductivity*, vol. 27, no. 4, pp. 1–9, 2017.

[8] S. K. Tolpygo and V. K. Semenov, "Increasing integration scale of superconductor electronics beyond one million josephson junctions," *Journal of Physics: Conference Series*, vol. 1559, no. 1, p. 012002, jun 2020. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1559/1/012002

[9] I. Nagaoka, M. Tanaka, K. Inoue, and A. Fujimaki, "A 48ghz 5.6 mw gate-level-pipelined multiplier using single-flux quantum logic," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 460–462.

[10] F. Zokaee and L. Jiang, "SMART: A heterogeneous scratchpad memory architecture for superconductor sfq-based systolic cnn accelerators," *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, Oct 2021. [Online]. Available: http://dx.doi.org/10.1145/3466752.3480041

[11] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aaa8415

[12] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing*, vol. 459, pp. 249–289, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221006706

[13] J. Han, L. Xu, M. M. Rafique, A. R. Butt, and S.-H. Lim, "A quantitative study of deep learning training on heterogeneous supercomputers," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 1–12.

[14] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–15. [Online]. Available: https://doi.org/10.1145/3341301.3359646

[15] K. Ishida, I. Byun, I. Nagaoka, K. Fukumitsu, M. Tanaka, S. Kawakami, T. Tanimoto, T. Ono, J. Kim, and K. Inoue, "Supernpu: An extremely fast neural processing unit using superconducting logic devices," *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9251979

[16] P. Sun, Y. Wen, R. Han, W. Feng, and S. Yan, "Gradientflow: Optimizing network performance for large-scale distributed dnn training," *IEEE Transactions on Big Data*, vol. 8, no. 2, pp. 495–507, 2022.

[17] I. Hounie, J. Elenter, and A. Ribeiro, "Neural networks with quantization constraints," 2022.

[18] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," 2017.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[20] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[25] Y. Lu, B. He, X. Tang, and M. Guo, "Synergy of dynamic frequency scaling and demotion on dram power management: Models and optimizations," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2367–2381, 2015.

[26] K.-D. Hwang, B. Kim, S.-Y. Byeon, K.-Y. Kim, D.-H. Kwon, H.-B. Lee, G.-I. Lee, S.-S. Yoon, J.-Y. Cha, S.-Y. Jang, S.-H. Lee, Y.-S. Joo, G.-S. Lee, S.-S. Xi, S.-B. Lim, K.-H. Chu, J.-H. Cho, J. Chun, J. Oh, J. Kim, and S.-H. Lee, "A 16gb/s/pin 8gb gddr6 dram with bandwidth extension techniques for high-speed applications," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 210–212.

[27] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 1, jun 2017. [Online]. Available: https://doi.org/10.1145/3084447

[28] R. Kashima, I. Nagaoka, M. Tanaka, T. Yamashita, and A. Fujimaki, "64-ghz datapath demonstration for bit-parallel sfq microprocessors based on a gate-level-pipeline structure," *IEEE Transactions on Applied Superconductivity*, vol. 31, no. 5, pp. 1–6, 2021.

[29] M. M. Islam, S. Alam, M. S. Hossain, K. Roy, and A. Aziz, "A review of cryogenic neuromorphic hardware," *Journal of Applied Physics*, vol. 133, no. 7, p. 070701, 2023. [Online]. Available: https://doi.org/10.1063/5.0133515

[30] P. Tschirhart and K. Segall, "Brainfreeze: Expanding the capabilities of neuromorphic systems using mixed-signal superconducting electronics," *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.750748

[31] M. Schneider, E. Toomey, G. Rowlands, J. Shainline, P. Tschirhart, and K. Segall, "Supermind: a survey of the potential of superconducting electronics for neuromorphic computing," *Superconductor Science and Technology*, vol. 35, no. 5, p. 053001, mar 2022. [Online]. Available: https://dx.doi.org/10.1088/1361-6668/ac4cd2

[32] H. He, Y. Yamanashi, and N. Yoshikawa, "Design of discrete hopfield neural network using a single flux quantum circuit," *IEEE Transactions on Applied Superconductivity*, vol. 32, no. 4, pp. 1–4, 2021.

[33] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

[34] ——, "Sparse distributed memory and related models," Tech. Rep., 1992.

[35] M. J. Flynn, P. Kanerva, and N. Bhadkamkar, "Sparse distributed memory: Principles and operation," Tech. Rep., 1989.

[36] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, 2017.

[37] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, and J. M. Rabaey, "Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 5880–5898, 2018.

[38] 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. [Online]. Available: https://arxiv.org/abs/2106.05268

[39] B. Emruli, R. W. Gayler, and F. Sandin, "Analogical mapping and inference with binary spatter codes and sparse distributed memory," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–8.

[40] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations," *ACM Comput. Surv.*, vol. 55, no. 6, dec 2022. [Online]. Available: https://doi.org/10.1145/3538531

[41] R. Vdovychenko and V. Tulchinsky, "Sparse distributed memory for sparse distributed data," in *Intelligent Systems and Applications*, K. Arai, Ed. Cham: Springer International Publishing, 2023, pp. 74–81.

[42] D. Kleyko, D. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges," *ACM Comput. Surv.*, vol. 55, no. 9, jan 2023. [Online]. Available: https://doi.org/10.1145/3558000

[43] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 64–69. [Online]. Available: https://doi.org/10.1145/2934583.2934624

[44] M. Kang, E. P. Kim, M.-s. Keel, and N. R. Shanbhag, "Energy-efficient and high throughput sparse distributed memory architecture," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 2505–2508.

[45] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 64–69. [Online]. Available: https://doi.org/10.1145/2934583.2934624

[46] E. Hassan, Y. Halawani, B. Mohammad, and H. Saleh, "Hyper-dimensional computing challenges and opportunities for ai applications," *IEEE Access*, pp. 1–1, 2021.

[47] A. A. Khan, S. Ollivier, S. Longofono, G. Hempel, J. Castrillon, and A. K. Jones, "Brain-inspired cognition in next generation racetrack memories," *ACM Trans. Embed. Comput. Syst.*, mar 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3524071

[48] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–8.

[49] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[50] V. Miranda and O. d'Aliberti, "Hyperdimensional computing encoding schemes for improved image classification," in *2022 IEEE International Symposium on Technologies for Homeland Security (HST)*, 2022, pp. 1–9.

[51] D. Liang, J. Shiomi, N. Miura, and H. Awano, "Distrihd: A memory efficient distributed binary hyperdimensional computing architecture for image classification," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 43–49.

[52] M. Imani, X. Yin, J. Messerly, S. Gupta, M. Niemier, X. S. Hu, and T. Rosing, "Searchd: A memory-centric hyperdimensional computing with stochastic training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2422–2433, 2020.

[53] Y. Ni, M. Issa, D. Abraham, M. Imani, X. Yin, and M. Imani, "Hdpg: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1141–1146. [Online]. Available: https://doi.org/10.1145/3489517.3530668

[54] P. Poduval, Z. Zou, X. Yin, E. Sadredini, and M. Imani, "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 781–786.

[55] A. Hernández-Cano, C. Zhuo, X. Yin, and M. Imani, "Reghd: Robust and efficient regression in hyper-dimensional learning system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 7–12.

[56] U. Quasthoff, M. Richter, and C. Biemann, "Corpus portal for search in monolingual corpora," in *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. Genoa, Italy: European Language Resources Association (ELRA), May 2006. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2006/pdf/641_pdf.pdf

[57] P. Koehn, "Europarl: A parallel corpus for statistical machine translation," in *Proceedings of Machine Translation Summit X: Papers*,

Phuket, Thailand, Sep. 13-15 2005, pp. 79–86. [Online]. Available: https://aclanthology.org/2005.mtsummit-papers.11

[58] V. Kaplunenko, M. Khabipov, D. Khokhlov, A. Kirichenko, V. Koshelets, and S. Kovtonyuk, "Experimental implementation of sfq ndro cells and 8-bit adc," *IEEE Transactions on Applied Superconductivity*, vol. 3, no. 1, pp. 2662–2665, 1993.

[59] "Wrspice circuit simulator," accessed: 2023-03-28. [Online]. Available: http://www.wrcad.com/wrspice.html

[60] "Superconducting integrated circuits," accessed: 2023-03-28. [Online]. Available: https://www.ll.mit.edu/research-and-development/advanced-technology/microsystems-prototyping-foundry/superconducting

[61] J. Clow, G. Tzimpragos, D. Dangwal, S. Guo, J. McMahan, and T. Sherwood, "A pythonic approach for rapid hardware prototyping and instrumentation," *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8056860

[62] D. Kirichenko, S. Sarwana, and A. Kirichenko, "Zero static power dissipation biasing of rsfq circuits," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. 776–779, June 2011. [Online]. Available: https://ieeexplore.ieee.org/document/5688194

[63] N. K. Katam, O. Mukhanov, and M. Pedram, "Simulation analysis and energy-saving techniques for ersfq circuits," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–7, 2019.

[64] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, "Energy-efficient superconducting computing—power budgets and requirements," *IEEE Transactions on Applied Superconductivity*, vol. 23, no. 3, pp. 1 701 610–1 701 610, 2013.

[65] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[66] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE transactions on Computers*, vol. 31, no. 03, pp. 260–264, 1982.

[67] G. Tzimpragos, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Volk, J. Shalf, and T. Sherwood, "A computational temporal logic for superconducting accelerators," ser. ASPLOS, 2020.