Generic architectures for efficient Hyper-Dimensional Computing

Sohum Datta

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-54
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-54.html

May 1, 2023

Generic architectures for efficient Hyper-Dimensional Computing

by

Sohum Datta

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jan Rabaey, Chair
Professor John Wawrzynek
Professor Bruno Olshausen
Professor Sayeef Salahuddin

Summer 2022

Abstract

Generic architectures for efficient Hyper-Dimensional Computing

by

Sohum Datta

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Jan Rabaey, Chair

The last decade has witnessed a slowdown in technology scaling. At the same time, the emergence of machine learning has substantially increased computational demand. While these trends have seriously challenged traditional paradigms for digital design, novel computing methods based on randomness can be leveraged for continued increase in performance and energy efficiency. Hyper-dimensional Computing (HDC), a brain-inspired paradigm using high-dimensional random vectors as its fundamental data-type, shows promise. It is known to provide competitive accuracy on sequential prediction tasks with far smaller model size and training time compared to conventional machine learning, and is robust to representation errors.

This dissertation considers efficient architectures for real-time Hyper-dimensional Computing on edge devices. After a study of implementations on classical compute platforms, a highly-pipelined, data-flow architecture is developed from the first principles of HDC. A detailed construction for a reasonably large HDC processor, capable of supporting a wide variety of prominent body-sensing applications, forms the foundation of this work.

The basic architecture is extended to HDC processors requiring more than a single bit of representation for vector elements. Using results from high-dimensional probability theory, a numerical normalization is proposed and its effectiveness is proven for applications obeying reasonable assumptions on vector elements' distribution. Verification experiments indicate that empirical performance of the proposed normalization is far better than the theoretical guarantee.

A 2048-bit wide HDC processor, designed using the architectural principles developed here, was manufactured in a leading technology node. The chip reliably achieves an energy cost of approximately 30 nanojoules per classification on a state-of-the-art body-sensing application – the best when compared to all known previous hardware in the existing literature. Measurements establish energy efficiency and robustness of the designed processor.

The architecture, arguments, experiments and measurements presented in this dissertation confirm the great potential of Hyper-Dimensional Computing as a computing paradigm capable of competitive performance in extremely energy-constrained environments.

Acknowledgements

This project would not be possible without the continuous support and encouragement from my adviser, Prof. Jan M. Rabaey. While working with him, I have learned the value of principled thinking, originality and perseverance. Although there have been great highs and tremendous lows throughout this journey of about seven years, Jan's support was instrumental in eventually effecting progress every time. Indeed, it has been a privilege to watch an *excellent* researcher think, brain-storm and make decisions so closely in over two dozen presentations, more than eighty group meetings and many more conversations – for which I am eternally grateful.

I have been very fortunate to have known Professors Bruno Olshausen, John Wawrzynek and Sayeef Salahuddin. Their careful examination of this work's arguments, in-depth discussions and specific feedback has been essential for me to successfully correct logical faults, improve my hypotheses and presentation of the material. Pentti Kanerva and Bruno have been foundational in forming and maintaining the Berkeley community interested in HDC, located at the Redwood institute. I'm very grateful for Prof. Subhasish Mitra's pithy comments and feedback on the subject and allied fields. I'm also pleasantly reminded of the five continuous hours on a bright afternoon at Stanford, where an extempore, free-for-all discussion had broken out in Subhasish's group – barely hinged to my presentation.

I am grateful for the outstanding teachers who have taught me during my stay at Berkeley. Memorable experiences include Krste Asanovic's and John Wawrzynek's expositions on classical architectural monuments and recent creativity, Jan's renowned sandbox for students to try their skills in imagining tiny processors (or its lack thereof); Bin Yu's brilliant and unbelievably informal course on statistics, augmented with mathematical treatments by Martin Wainwright, William Fithian and Michael Jordan (in the order of booming complexity), and the entirely new experience at Tom Griffith's seminars on cognition. I am especially grateful to Jonathan Shewchuk for the memorable and unconventional introduction to machine learning in Spring 2016, and for the wonderful opportunity to teach and influence that flagship undergraduate and early-graduate course in Computer Science (with more than 650 completing students). Jonathan also gave me the *rare* opportunity to write all questions for an entire homework set and an essay-type written question for the final exam during both semesters that I assisted in teaching. Teaching and interacting with such a large and diverse group of students has been an incredible learning experience.

I have been fortunate to work with brilliant colleagues during this period. Abbas Rahimi (IBM Zurich) and Mohsen Imani (UC Irvine) have been very helpful in guiding me through the proverbial research thicket by sharing data sets, tips and producing a stream of related research – guiding me to my destination. Andy Zhou, Ali Moin and George Alexandrov's splendid data set and paper on EMG hand-gesture recognition provided a great final benchmark for comparison. Paxon Frady, Friedrich Sommer and Denis Kleyko's papers were the first theoretical works on the subject that I came across after reading Pentti's decades-old book. Finally, I especially appreciate Alisha Menon, Youbin Kim, Laura Galindez, Denis Kleyko, Spencer Kent, Matthew Andersen and Mohamed Ibrahim for their thoughtful comments on my work during the dozens of group presentations over the years. During my research at Berkeley, I was fortunate to work with visitors from University of the Philippines Diliman. Ryan A. G. Antonio and Aldrin R. S. Ison were very helpful in modifying and deploying my source code on an embedded GPU and CPU core. They were quick learners, and our collaboration produced the first publication of this project. Denise Soriano, Alexis Czezar Cruz Torreno and Bentz Del Mundo visited Berkeley in early 2020 – while the pandemic was rapidly spreading in North America. Despite that, they collected a great amount of useful data that helped me scope-out a successful idea later.

The chip taped-out during this project would not have been possible without Brian Richards. Apart from his deep expertise, excellent design suggestions and incisive questions during technical reviews, I will miss his usual cool-headed demeanor in the presence of tape-out deadlines and him always cheering me up when I'm falling behind or getting demotivated. Brian is a tremendous resource for all graduate students at BWRC, and the generations of chip designers taping out and graduating from my lab are privileged to have such excellent research staff to support them. The graduate students who created the `Hammer` tape-out tool, especially Harrison Liew, were essential for making physical design manageable. Harrison was available and super helpful during the entire eight-months long physical design phase, slowly teaching me how to use `Hammer` and providing guidance when I got stuck.

I'm thankful to Youbin Kim, Daniel Sun and Mohamed Ibrahim for lending me a helping hand at times when the effort seemed unbearable. Youbin assisted me in completing the chip's Input/Output pad-frame, Daniel was instrumental in getting a `Hammer` setup started for my chip, and Mohamed was very helpful in fixing a few last-moment DRC errors in my design.

The enormous effort spent in designing and fabricating the chip would have been in vain without a successful testing setup. Crucial to that is to design a Printed-Circuit Board (PCB) that can properly support tests and measurements. My first PCB design experience wouldn't have been successful without the continuous advise of Brian and James Dunn. They are experts in PCB design, and their periodic reviews were essential for debugging within deadlines. James was generous with his time in helping me with soldering certain components to the board. Anita Flynn and Robert Kondner *selflessly* devoted their time to my project outside of normal working hours – helping my board by designing PCB foot-prints for some non-standard package components. Anita's attention to detail and careful documentation stood out as great skills that I should cultivate to become an excellent engineer.

The Berkeley Wireless Research Center (BWRC) is lucky to have superb administrative staff such as Columba Candy Corpus-Steudeman, Mikaela Cavizo-Briggs and Yessica Bravo. Candy, assisted by Yessica and Mikaela, has been extremely kind and supportive throughout this journey – handling all research, financial and travel-related paperwork with great organization. Jeffrey Anderson-Lee helped me stay connected to the lab machines even while the pandemic was spreading and lock-downs were being announced. Most importantly, I want thank Candy for greeting me with a *smile everyday* at the reception area near the entrance all these years.

I'm indebted to the Taiwan Semiconductor Manufacturing Company, Limited (TSMC) for funding my research, providing us with the physical design kit and manufacturing my chip in an industry-leading technology node. I am thankful to the Semiconductor Research Corporation (SRC), the Defense Advanced Research Project Agency (DARPA), the National Science Foundation (NSF), TSMC and all sponsors of the Berkeley Wireless Research Center (BWRC) at UC Berkeley for funding and supporting my project.

Finally, I thank the wider community for expressing an interest in my research, providing counter-arguments and sharing relevant information from their fields related to my project. They include Jared Zerbe (Apple), Bryan Raines (Apple), Dana Massie (Apple), William Athas (Apple), Julia Ng (NVIDIA), Jonathan Yedidia (Analog Garage), Nicolas Le Dortz (Analog Garage), Phillip Nadeau (Analog Garage), Mehul Tikekar (Waymo), Chiraag Juvekar (Apple), Tony Wu (Meta), Haitong Li (Purdue), Geoffrey Burr (IBM), Matthew Ziegler (IBM), Krishnan Kailas (IBM), Mondira Pant (Intel), Mike Davies (Intel), Christopher Hughes (Intel), Jinjun Xiong (IBM), Todd Younkin (SRC), Ramesh Chauhan (Qualcomm), Cliff Young (Google) and Derrick Aguren (AMD). Discussions with the larger community expanded my horizons and allowed to gain a longer-term perspective on my work.

Berkeley, CA Sohum Datta

This dissertation is dedicated to my father.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Two crucial events in the last decade determine the pace of technical innovation today. The first is the gradual slowdown in relentless miniaturization of semiconductor devices, known as Moore's law [1]. Beginning in the late 1960s, the self-aligned, planar-gate silicon metal-oxide transistor created the foundation for a three-decade period of relentless and exponential progress in transistor miniaturization. Gordon Moore's prediction of transistor counts doubling every two years was remarkably prescient for this era, aided by Robert Dennard's scaling prescriptions for reliably manufacturing faster and smaller transistors [2].

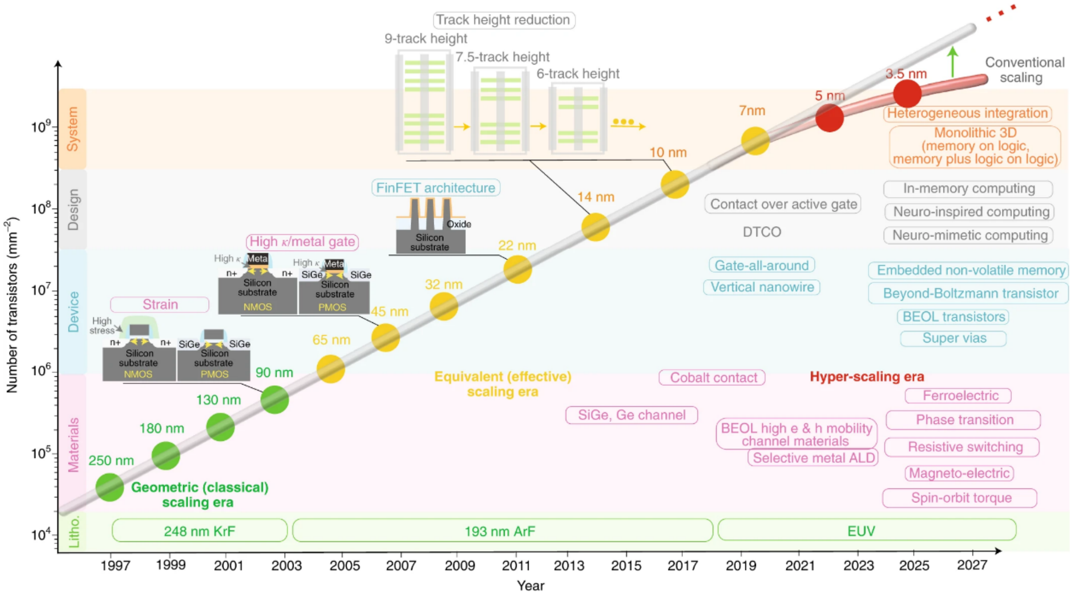


Figure 1-1: Figure 1 of [3]: eras of transistor scaling.

At turn of the millennium, this classical period of geometric scaling of transistors gave way to various manufacturing innovations which *equivalently* produced the same effect as the (slowing) rate of physical dimension scaling (see figure 1-1). It is believed that the current period of equivalent scaling of transistors using novel gate materials and geometries is likely to give way to hyper-scaling i.e. functionality-aware beyond-Boltzmann transistors after 2025 [3]. And as transistor dimensions approach 10nm, variability and reliability effects begin to dominate its deterministic behavior [4]. For continued miniaturization, new avenues of research into materials, semiconductor physics and organic chemistry for emerging devices have materialized [5].

Secondly, the rise of data-driven learning algorithms have completely changed the way businesses function [6]. Due to the widespread proliferation of sensory devices and improvements in connectivity, the huge amounts of data gathered must be processed for ensuring quality of services. Furthermore, mobile devices (e.g. smartphones, tablets, sensor-nodes in sensor networks) function under limited bandwidth, battery and storage capacity, thereby requiring high energy efficiency in their computations [7].

Clearly, one way to harness the two trends going forward is to perform machine learning on emerging post-Moore devices with much lower energy footprints. This is especially useful for edge-based Internet-of-Things (IoT), where data is partially processed immediately after collection to reduce bandwidth usage and server workload. Emerging devices allow such computations to meet the strict energy constraints required. However, adapting emerging devices to the exact-computing paradigm is difficult due to their inherent variability [8] . As energy efficiency no longer scales with integration capacity, voltage reduction and near-threshold operation reduces power consumption at the expense of favorable signal-to-noise ratio (SNR) [9]. Finally, for conventional architectures such as CPUs and General-Purpose GPUs (GPGPUs), few applications today (including data mining and classification) have enough parallelism to completely utilize available hardware [10].

While challenges of using unreliable components have long been known [11], biology offers the most concrete inspiration. For example, our brain processes massive data (3.6×10^{15} synaptic ops./s) with very slow and diverse neurons (typical firing rates are 10 - 100 Hz) while exhibiting tremendous energy efficiency (total power is about 12 W) [12]. Consequently, **brain-inspired computing** could provide the required robustness and scalability for continued improvements.

Hyper-Dimensional Computing (HDC) is one such nano-scalable paradigm [14], and is known to excel in body-based sensing/IoT applications [15]. Also known as Vector Symbolic Architectures (VSAs), it originated from a theoretical model of cognitive reasoning [16, 17]. It is motivated by the model that brains compute by transforming activation patterns of a *large population* of neurons (see figure 1-2 and the adjoining explanation). Hence, tolerance to variability is inherent: changes in activation of a few neurons do not affect the overall functionality. Its energy efficiency and robustness to noise (introduced by reduced supply voltage VDD) in the data path was demonstrated for language recognition [18, 19] and tested on fabricated systems based on emerging devices: a hybrid of carbon nanotube field-effect transistors (CNFETs) and resistive RAM (RRAM) memory in [20], and a CMOS/vertical-RRAM (VRRAM) implementation in [21]. [21] also demonstrated the robustness of HDC to inherent RRAM variability in endurance cycles and wafer-level device-to-device characteristics. Buoyed by the potential benefits offered by HDC, research has grown dramatically over the past few years (as shown in figure 1-3).

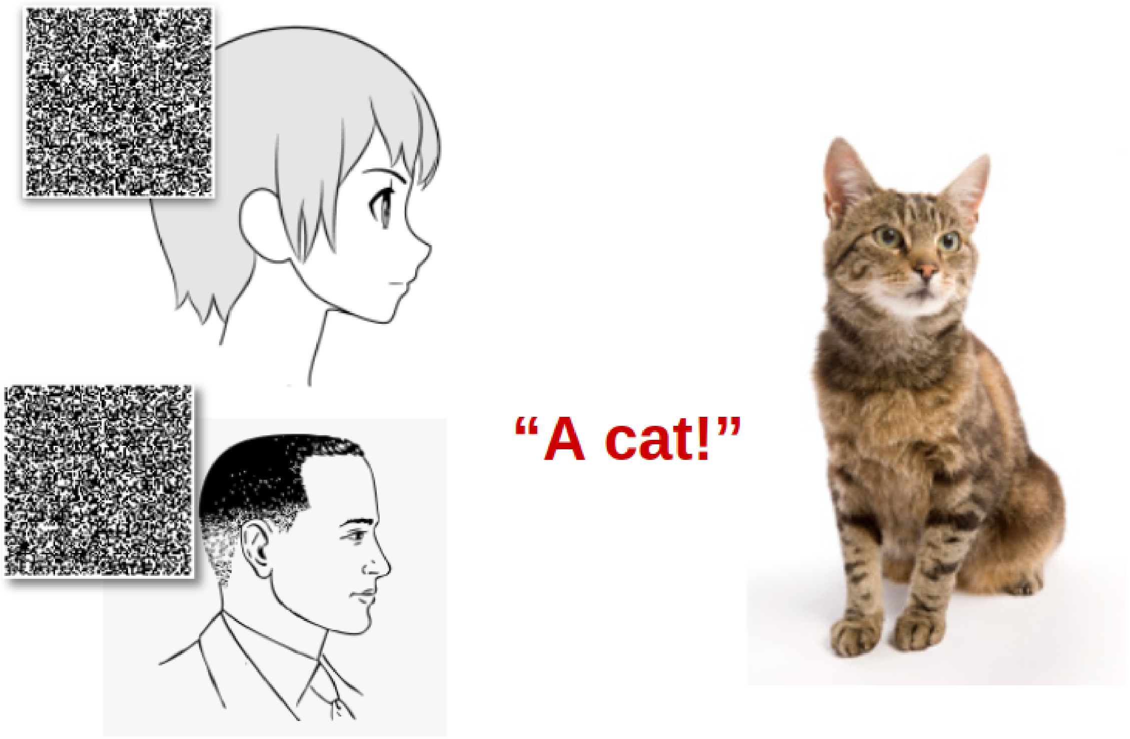


Figure 1-2: Neural-sampling hypothesis and Hyper-Dimensional Computing.

The neural sampling hypothesis [13] is a commonly-used model for neural computation and human cognition. The brain is thought to perform computations using probability distributions by transmitting neural activity samples between neurons, whereby the activity of a large population of neurons *directly represent samples from the said distributions*.

Consequently, even though two individuals may have *completely different and random neural activations* (illustrated here as a display of random noise) to a common stimulus (illustrated here as an image of a cat) due to the distinct ways in which their nerves and sensory tissues are interconnected in the sensory organs, the subsequent computations in their brains conducted by sampling, transforming and transmitting these sensory activations still results in identical and predictable behaviour (i.e. both identify the image as “a cat”).

Hyper-Dimensional Computing represents the sensory activations in individuals as binary random vectors with *large number of bits* and abstracts the subsequent transformations as a sequence of operations from a collection of operators defined on large binary vectors.

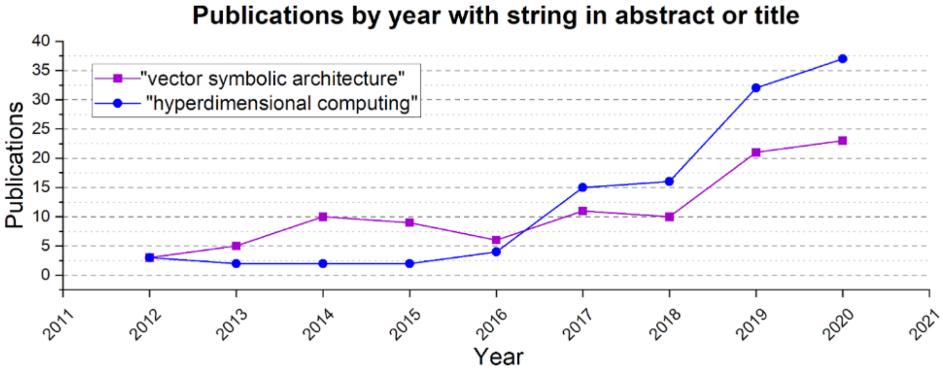
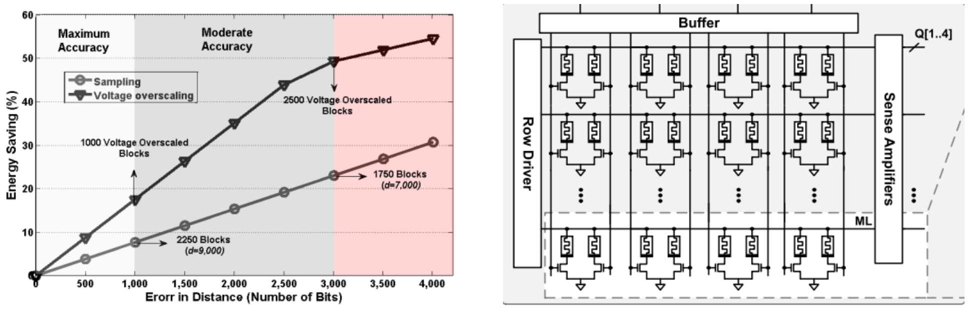


Figure 1-3: Research literature related to HDC is increasing rapidly with time. Plot produced using data retrieved from `dimensions.ai`.



(a) Figure 5 of [18]: Energy savings in Associative Memory by sampling and distributed VDD overscaling

(b) Figure 3(b) of [18]: resistive Content Addressable Memory(CAM) arrays used in HDC associative search

Figure 1-4: HDC is robust to representation errors and well-suited for in-memory hardware implementations.
(a) As shown in [18], robustness can be utilized by trading off tolerable representation errors for lower energy consumption. (b) Furthermore, the distributed and simple nature of HDC computations (especially associative search) makes it ideal for computing directly in or near memory cells.

However, as mentioned in section 2.4 in the next chapter, most research on HDC hardware are limited to specific datasets or applications and often have low data-width (eg. 32 bits on-chip to compute hyper-vectors containing thousands of bits in [20]), requiring large amount of time-multiplexing to simulate the complete machine of full width. Data-paths specific to applications other than language recognition have also been proposed [22], but a general HDC system is yet to be developed.

A general and widely programmable HDC architecture, which can be easily programmed to perform a variety of applications on different datasets, is crucial for evaluating HDC as a viable IoT and in-sensor computing paradigm [15]. This requires developing the fundamental architectural blocks that are configured and interconnected to produce a complete system. A comprehensive exploration of the above is the main goal of this work.

Chapter 2

Hyper-Dimensional Computing : preliminaries and a survey

Hyper-Dimensional Computing (HDC) emerged from a theoretical model of memory and cognition [17]. It is based on the fact that human brains compute by transforming activation patterns of a large mass of neurons. The set of activations are modeled as points in very high dimensional spaces ($d \geq 1000$), and neural processing as transformations in this space. The central idea is that the mathematical properties of high-dimensional metric spaces – which pose a challenge to common machine learning algorithms [23] such as nearest-neighbor search [24], clustering [25] and regression [26] – can be used to explain cognitive functions like association of concepts, learning and recalling by analogy. Simple operations such as superposition, binding, permutation and their inverses form an algebraic field, giving (in principle) the same universality as algebra with numbers [27].

The HDC formalism can also be regarded as a mathematical abstraction of the memory functions exhibited by the human brain, similar to McCullouch and Pitts' *artificial neurons* formulated in 1943 [28]. There are numerous paradigms of similar or directly related origin, such as Holographic Reduced Representation [27], Binary Spatter Code [29] and Semantic Pointer Architecture [30]. Collectively, these models of human cognition are referred to as **Vector Symbolic Architectures (VSAs)** in the wider literature related to Psychology and Cognitive Neuro-science [31].

While there are several variants of HDC suited for different applications [14], this chapter provides the necessary preliminaries for the **Multiply-Add-Permute** (MAP) architecture most commonly used in the HDC hardware community [32]. The MAP architecture is the only HDC variant studied in this dissertation. After a brief introduction to the basic concepts and a demonstration of its use for two exemplar applications, a survey of published literature related to hardware designs capable of supporting multiple HDC algorithms concludes this chapter.

2.1 Orthogonality in high dimensions

Hyper-dimensional Computing defines random high-dimensional vectors ($d > 1000$) as its fundamental data type [16, 14]. It is a **holographic** computing framework: unlike arithmetic over numbers, no vector component contains more information than any other. Although vectors with elements from any algebraic field can be used (see Table I of [14]), we will consider only binary vectors as it results in the simplest hardware.

To compare vectors, a distance metric is required. Hamming distance (denoted by $d_H(a, b)$) is the number of dissimilar elements between vectors a and b. Two binary vectors x and y of dimension d are said to be orthogonal if $d_H(x, y) = d/2$. This definition is more familiar in bipolar code (0-valued elements replaced by integer -1): orthogonal x and y have zero inner product, $\langle x, y\rangle = 0$.

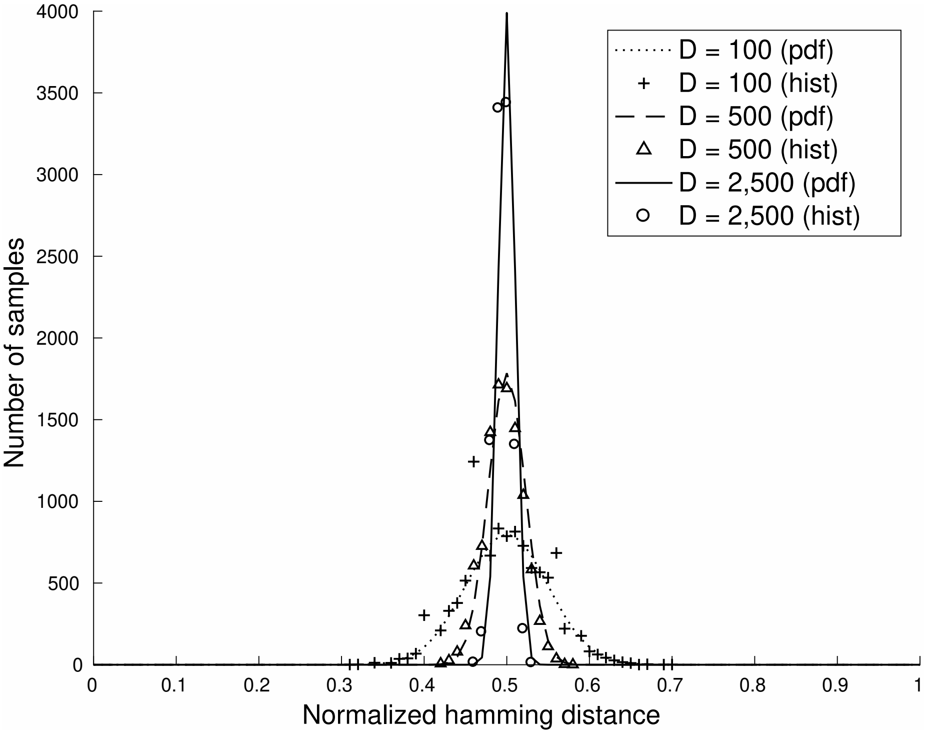


Figure 2-1: Orthogonality in High Dimensions.
Shown histogram (hist) and scaled probability density function (pdf) of normalized hamming distances for $10,000$ pairs of random binary vectors with varying dimension d. For ease of plotting, the normal approximation of the binomial probability distribution is used. All vectors are generated uniformly and distance is normalized by d. Note the sharper concentration around 0.5 as vector dimension d increases.

The underlying principle of HDC is *almost certain orthogonality* in high-dimensional spaces. This is also called "almost-sure orthogonality", "pseudo-orthogonality", "near-orthogonality" and "orthogonality in high dimensions" in the literature. For a rigorous demonstration, see that if vectors x and y are chosen independently and uniformly from $\{0, 1\}^d$ (i.e. probability of any bit being 1 is $p = 1/2$), their hamming distance is binomially distributed: $d_H(x, y) \sim Bin(d, p = 1/2)$. Fig. 2-1 plots a histogram (hist) of

$d_H(x, y)$ normalized by dimension d for $10,000$ randomly-generated pairs (x, y). It also plots the density function (pdf) for Normal Approximation $\mathcal{N}(dp, dp(1-p))$ of $Bin(d, p)$ scaled to have an area equalling sample size $10,000$. The Normal Approximation helps in plotting and is very accurate for high dimensions: using the Berry-Essen bound (Theorem 10.4 in [33]) for $X \sim Bin(d, p),\ Y \sim \mathcal{N}(dp, dp(1-p)),\ p = 1/2$ we have that $\forall t \in \mathbb{R}$:

$$|\Pr(X \leq t) - \Pr(Y \leq t)| \leq \frac{4 - 8p(1-p)}{5\sqrt{dp(1-p)}}$$

Using $p = 1/2$ and for dimension $d \geq 1024$, the maximum error in cumulative distribution $(max_{0<t<1}|\Pr(X \leq t) - \Pr(Y \leq t)|)$ is as most 0.025.

Given that the hamming distance between random vectors is binomially distributed, it can be shown that (Theorem 1 of [34]):

$$\Pr\left[\left|\frac{d_H(x, y)}{d} - \frac{1}{2}\right| \geq \epsilon\right] < 2e^{-2d\epsilon^2} \tag{2.1}$$

Only high dimensions ($d > 1000$) result in a meaningful right-hand side in equation 2.1 [35]. Then random vectors x and y have normalized distance very close to 0.5. The *exponential* drop in probability beyond ϵ-deviation from the mean is the crucial property exploited in HDC algorithms. This phenomenon is a consequence of concentration of functions in high-dimensional geometry. For the interested reader, [36] provides an excellent treatment from a non-asymptotic viewpoint. Chapter 5 uses related phenomena to develop efficient architectures for HDC.

All high-dimensional vectors used in a given computation will be called **hypervectors**. When the context is obvious, hyper-vectors and vectors will be used interchangeably.

2.2 The Multiply-Add-Permute (MAP) paradigm of Hyper-Dimensional Computing

To simplify the formulation of HDC operations, the following conventions will be adopted throughout the dissertation:

- The starting hyper-vectors are binary and generated uniformly at random.
- Unless stated otherwise, bipolar hyper-vectors are adopted instead of binary hyper-vectors, where -1 replaces 0 as the hyper-vector element. The main advantage of this adoption is that superimposed bipolar hyper-vectors (defined below) preserves the elements' mean value of 0.

In addition to high dimensionality, a set of operations are required that preserve near orthogonality. Although there are many sets of operations with equivalent performance (see Table I of [14]), the **Multiply-Add-Permute (MAP)** framework is most suitable as it uses *binary* hyper-vectors – the easiest to implement in digital logic. The following operations are fundamental to the MAP paradigm:

- **Multiplication/Binding** is useful for forming *associations* among related hyper-vectors. X and Y are bound together to form $C = X \oplus Y$ that is nearly orthogonal to both its constituents. It is implemented as element-wise XOR $x_i{}^\wedge y_i$ of two binary hyper-vectors, and as the negative of the product $-x_i y_i$ of two bipolar hyper-vectors.
- **Addition/Superposition** is the primary *conjunctive* operation. Based on Hebbian learning [27], the goal is to find a hyper-vector z representing the set of operand hyper-vectors $\{x_1, x_2, ...x_n\}$. It is denoted by $z = x_1 + x_2 + ... + x_n$ and implemented by performing element-wise integer sum of operand hyper-vectors.
- **Thresholding** each element of a hyper-vector X at the element-wise mean (0 for bipolar) is denoted by $z = [X]$. An element of the thresholded hyper-vector is $+1$ if the corresponding element of X is non-negative; it is -1 otherwise. Note that thresholding is idempotent i.e. $[X] = X$ for all bipolar vectors X.
  Thresholding is almost always performed after superposition of multiple bipolar hyper-vectors $[x_1 + x_2 + \ldots x_n]$ so that the output hyper-vector is bipolar, formed by element-wise majority of the superimposed hyper-vectors $x_1, x_2 \ldots x_n$.
- **Permutation** is a unary operation such that the permuted hyper-vector denoted by $\rho(x)$ is nearly orthogonal to its operand x. The choice of permutation does not matter as long is it has no fixed points – so that the period is the largest value possible: the HDC dimension d. Circular shift (in either direction) is a widely adopted permutation. Hyper-vector x permuted n times is written $\rho^n(x)$.

Associative search. As Fig. 2-1 shows, it is very rare for random hyper-vectors to deviate much from orthogonality. The addition operation generates non-orthogonal hyper-vectors from random operand hyper-vectors. This allows one to capture semantics between hyper-vectors as any significant deviation from orthogonality among two hyper-vectors implies some dependency. In the final stage of *all HDC algorithms*, an **associative search** is performed among all possible solution hyper-vectors to find the closest match to the query hyper-vector calculated from input bipolar hyper-vectors using MAP operations.

Following from their definitions [16], the four fundamental MAP operations in HDC have the following laws governing the relationships among them:

1. Superposition and binding are commutative: for all vectors A, B we have $A+B = B+A$ and $A \oplus B = B \oplus A$.
2. Superposition and binding are associative: for all vectors A, B, C we have $(A+B)+C = A+(B+C)$ and $(A \oplus B) \oplus C = A \oplus (B \oplus C)$.
3. Binding can be inverted: for all vectors A and B we have $A \oplus A \oplus B = B$
4. Permutation distributes over binding: for all integers $n \geq 1$, the following holds true for all vectors A, B: $\rho^n(A \oplus B) = \rho^n(A) \oplus \rho^n(B)$.
5. Permutation distributes over superposition and thresholding: for all integers $n \geq 1$, the following holds true for all vectors A, B: $\rho^n(A+B) = \rho^n(A) + \rho^n(B)$, and for all vectors C we have that $\rho^n([C]) = [\rho^n(C)]$.
6. Binding distributes over superposition and thresholding: For all vectors A, B, C the following hold: $A \oplus [B+C] = [(A \oplus B) + (A \oplus C)]$ and $A \oplus [C] = [A \oplus C]$.

2.3 Examples of computing with hyper-vectors

HDC has been employed in a range of applications in supervised classification [37]. Prominent examples include human-sensing and biomedical signal processing [38, 39], classification using multimodal sensor fusion [40] and DNA pattern matching [41, 42]. Other notable works in the wider application space include using hyper-vectors to ensure trustable service discovery in highly decentralized and dynamic networks [43], predicting onset of short-term [44] and long-term [45] seizures in patients, performing factorization of integers using resonator networks [46], activity recognition of subjects by classifying their radar images [47], characterization of circuit reliability in state-of-the-art technology node [48], constructing a “proof-of-use” blockchain [49], electrocardiogram-based emotion recognition when subjects are in motion [50], guaranteeing privacy in distributed learning systems [51], using HDC and feed-forward neural network to recognize driving style for Advanced Driver Assistance Systems (ADAS) [52], building recommender systems [53], efficient communication via noisy channels and near-channel classification [54], classification of massive DNA methylation data for predicting cancer [55] and for dynamic vision sensing [56].

Several papers have proposed using principles of HDC such as orthogonality in high dimensions in other fields of study. Most prominently, proposals of hybrid HDC-neural network systems and machine learning algorithms with VSA-inspired learning cost functions have received attention. For example, in [57], a hybrid HDC-neural network system for storage and inference on knowledge graphs was proposed based on concentration of measures in high dimensions. HDC was used to implement the back-end processing pipeline in the vision engine for active perception in robots [58] and training multiple neural networks simultaneously in a single model [59]. In [60], a neuro-symbolic system constructed as a hybrid of neural networks and VSA was used to solve Raven’s progressive matrices (a common exercise present in Intelligence Quotient tests). Most recently, [61] used the orthogonality in high dimensions as a training objective to train Convolutional Neural Networks (CNN) that can learn hyper-vector representations of images and store it in a associative memory. The memory-augment CNN supported by orthogonality can dynamically expand to learn new image categories; experiments on large data sets of natural images show it performs better than state-of-the-art CNN models. Use of reinforcement learning to automatically search for the HDC algorithm that results in the best accuracy on a given classification task has also been proposed [62].

All applications of HDC use the symbolic nature of its computations. The idea is to use the $e^{-\Theta(t^2)}$ drop in probability beyond a t-deviation from the mean *normalized* distance between high-dimensional random vectors (as demonstrated in section 2.1) to encode semantic relationships. The following sub-section illustrates this with a few examples.

2.3.1 Encoding semantics with random hyper-vectors

Perhaps the foremost feature of HDC that distinguishes it from other algorithms is its symbolic nature of processing. HDC is related to symbolic models of computation – like connectionist models proposed and studied in late 1980s (the interested reader is referred to chapters $1 - 3$ of [27] for a detailed survey). Symbolic computation begins by defining **symbols** that represent entities from the universe where the reasoning is to be conducted. For HDC, a randomly generated hyper-vector is assigned to each such entity, and the collection of all entity-vector pairs is *stored for the entire duration of the computation.*

Set membership. The most fundamental semantic relationship is membership of entities in a collection. Using random hyper-vectors to represent entities and using superposition to represent a collection of entities, HDC provides an easy algorithm to test set membership. For example, suppose there are 26 entities which are represented by random (bipolar) hyper-vectors $A, B, \dots Z$. Then, to represent the set $\mathbb{S} = \{A, B, C\}$ one can simply superimpose the hyper-vectors of its members: $\mathcal{S} = [A + B + C]$.

To test membership of $A, B, \dots Z$ in $\mathbb{S}$, check whether the hamming distance between their representation hyper-vectors $\mathrm{d_H}$ is smaller than $d/2$ by a significant margin. Thus for non-members such as X, we have that $\mathrm{d_H}(X, \mathcal{S}) \approx d/2$ with very high probability because of orthogonality of random hyper-vectors in high dimensions. For members like A, we have $\mathrm{d_H}(A, \mathcal{S}) = \mathrm{d_H}(A, [A + B + C])$ which is close to $\mathrm{d_H}(A, A) = 0$ (see [16] for a demonstration).

Analogical reasoning. Using hyper-vectors as symbols also allows representing analogical structures and reasoning about them. For example, consider a system capable of answering "What is the Dollar of Mexico?". In order to get a reply to this question, it is necessary to infer the entity that bears the same relation to "Mexico" as "Dollar" does to "United States". If "United States" and "Mexico" are represented by random hyper-vectors $\mathtt{US}$ and $\mathtt{MEXICO}$ and their respective currencies by hyper-vectors $\mathtt{DOLLAR}$ and $\mathtt{PESO}$, then the hyper-vector $\mathtt{CURRENCY} \triangleq [\mathtt{US} \oplus \mathtt{DOLLAR} + \mathtt{MEXICO} \oplus \mathtt{PESO}]$ represents the collection of country-currency analogies for "United States" and "Mexico" [63]. In other words, the hyper-vector $\mathtt{CURRENCY}$ encodes the set of (unordered) pairs formed from the four entities “United States”, “Dollar”, “Mexico” and “Peso”:

$$\text{“Currency”} \triangleq \{(\text{“United States”}, \ \text{“Dollar”}), (\text{“Mexico”}, \text{“Peso”})\} \tag{2.2}$$

One may now compute the “Dollar” of “Mexico” by seeking entity X such that (“Mexico”, X) $\in$ “Currency” in equation 2.2. This can be solved as follows:

$$\begin{aligned}
&\mathrm{d_H}(X \oplus \texttt{MEXICO}, \texttt{CURRENCY}) \approx 0 \\
\Longrightarrow\ &\mathrm{d_H}(X \oplus \texttt{MEXICO} \oplus \texttt{MEXICO}, \texttt{CURRENCY} \oplus \texttt{MEXICO}) = \mathrm{d_H}(X, \texttt{CURRENCY} \oplus \texttt{MEXICO}) \approx 0 \\
\Longrightarrow\ &X = \underset{x \in \{\texttt{US},\texttt{MEXICO},\texttt{DOLLAR},\texttt{PESO}\}}{\arg\min} \mathrm{d_H}(x, \texttt{MEXICO} \oplus \texttt{CURRENCY})
\end{aligned} \tag{2.3}$$

Of the four entities defined so far, following from the properties of HDC operations in section 2.2 one can conclude that $X = \texttt{PESO}$. This deduction is shown below:

$$\begin{aligned}
&\mathrm{d_H}(X, \texttt{MEXICO} \oplus \texttt{CURRENCY}) \approx 0 \\
\Longrightarrow\ &\mathrm{d_H}(X, \texttt{MEXICO} \oplus [\texttt{US} \oplus \texttt{DOLLAR} + \texttt{MEXICO} \oplus \texttt{PESO}]) \approx 0 \\
\Longrightarrow\ &\mathrm{d_H}(X, [\texttt{PESO} + \texttt{MEXICO} \oplus \texttt{US} \oplus \texttt{DOLLAR}]) \approx 0 \\
\Longrightarrow\ &\mathrm{d_H}(X, [\texttt{PESO} + (\text{random hyper-vector})]) \approx 0 \\
\Longrightarrow\ &X = \texttt{PESO}
\end{aligned} \tag{2.4}$$

The final step in equation 2.4 above determines which of the four defined entities that are assigned random hyper-vectors in this problem (i.e. “United States”, “Dollar”, “Mexico” and “Peso”) is a member of the transformed set

$$\{\text{“Peso”}, (\text{“Mexico”}, \ \text{“United States”}, \ \text{“Dollar”})\}$$

Since triples are not a part of the four defined entities, “Peso” is the final solution.

Hyper-Dimensional Computing has been used to represent a variety of data structures such as lists, sequences, trees and Turing machines [27, 64]. The method of solving for the solution X consists of systematically transforming the set “Currency” to a new set of symbols by applying MAP operations from section 2.2 so that only the correct solution among all the recognized entities is its member. Maintaining the collection of assigned hyper-vectors representing all the *defined* entities, and associatively searching the transformed vector (like $\texttt{MEXICO} \oplus \texttt{CURRENCY}$) for the closest entity hyper-vector are common features of *all* HDC algorithms.

The next sub-section introduces language recognition using HDC operations and orthogonality among random hyper-vectors.

2.3.2 Language Recognition using HDC

Language recognition using HDC was a seminal work [65] that inspired a subsequent wave of research on using HDC for supervised classification tasks. This application also illustrates how the symbolic nature of HDC and orthogonality in high dimensions may be employed for a supervised classification task.

HDC language recognition was first performed on the EUROPARL corpus of 21 European languages transliterated in the Latin alphabet [19, 65]. The EUROPARL corpus was extracted from the proceedings of the European parliament. Due to its diplomatic nature, the sentences are longer and use larger vocabulary than in everyday use. Therefore, the EUROPARL corpus is considered to be a difficult dataset for language recognition and translation.

The n-gram baseline model. The goal is to recognize the language from a *short sentence* of text transliterated in the Latin alphabet. Written languages can be modeled as a probability distribution on character sequences or word sequences of finite length n, called n-grams [66]. While more sophisticated models such as dictionary of words, phrases, etc. can be used, studies indicate they increase model complexity with negligible gains [67, 68]. Since the language recognition algorithm is likely to be a front-end of a complex language-processing pipeline, having a complicated and large recognition model effectively nullifies the advantages of a simple and lightweight front-end. While training a language, raw n-gram frequency counts are generated from a large corpus from that language and iteratively smoothened [69] to remove outlier artifacts. The resulting n-gram distribution is the trained language model. The steps are repeated for a test query, and the trained model with the closest distribution is the language prediction.

HDC algorithm for n-grams. The first step is to assign random hyper-vectors to all meaningful entities in this task. Assigning hyper-vectors for each character of the Latin alphabet and a few punctuation marks should suffice. In [19] and [65], HDC dimension $d = 10000$ was used. Using smaller hyper-vectors with $d = 2048$ produces reasonable accuracy [32]. The HDC algorithm uses the assigned character hyper-vectors to encode the training text of a language and generate a *single hyper-vector* for each language.

A direct equivalent of frequency counting is the superposition of hyper-vectors representing each occurring n-gram in the text. Permutation and binding are used to generate an n-gram vector from constituent character hyper-vectors. For example, the 3-gram "abc" is encoded as $V_{abc} \triangleq \rho^2(V_a) \oplus \rho(V_b) \oplus V_c$, where V_z is hyper-vector representing character or character sequence z. From properties of binding and permutation in section 2.2, deduce that all distinct n-gram and character hyper-vectors are nearly orthogonal to each other.

Hence, the language hyper-vector is the thresholded superposition of all n-gram hyper-vectors from the training text. The test hyper-vector is computed similarly, and the language with closest hyper-vector (i.e. the smallest hamming distance) is returned as the prediction. Since the superimposed language vector is in the linear space spanned by the set of all n-gram vectors, the class with the closest n-gram distribution from baseline equivalently has the smallest distance in HDC.

HDC has an accuracy of *96.7 %* against a baseline of *97.1 %* [19] for $n = 4$ and HDC dimension $d = 10000$. However, it is an online algorithm requiring a *single iteration* though the entire training text. The deviations from orthogonality in high dimensions (in fig. 2-1) automatically smoothen the superimposed multi-set of n-grams, obviating the need for iterations over a possibly large training corpus. Finally, the HDC model size (1 vector/class) is fixed with n-gram size but *grows exponentially* in the baseline. Indeed, for $n = 4$, the HDC model is **20X smaller** than the baseline model.

2.4 A summary of hardware designs for Hyper-Dimensional Computing

Several hardware designs have been proposed in the Hyper-Dimensional Computing community. Following the demonstration of associative memory's robustness to noise (introduced by reduced VDD) in the HDC data path for language recognition [18, 19], a hybrid design of Carbon Nanotube Field-Effect Transistors (CNFETs) and Resistive RAM (RRAM) memory in [20] was fabricated and tested for EUROPARL language recognition. A CMOS/vertical-RRAM (VRRAM) implementation in [21] demonstrated the robustness of HDC to inherent RRAM variability in endurance cycles and wafer-level device-to-device characteristics. In [70], a Phase-Change Memory (PCM) based Associative Memory and Encoding module with digital CMOS peripherals was simulated for EUROPARL language recognition.

However, all HDC hardware designs *fabricated and measured* so far are tailor-made for a specific application or data set. Both [21] and [20] are hard-wired to perform only language recognition on a specific dataset; these chips cannot be used for other datasets and applications. The data path simulated in [70] and the chip illustrated in [71] can only perform a 2-min-term approximation of the complete n-gram in language recognition, and can only compute n-grams from a single data stream such as characters in a single text. Furthermore, its encoder contains CMOS digital logic which was simulated in software – only the PCM memory blocks were measured in hardware. Designs [21, 20] also have low data-width (32 bits), requiring large amount of time-multiplexing to simulate the complete hyper-vector in full width (> 1000 bits). Data paths specific to applications other than language recognition have also been proposed for future fabrication [22].

A great majority of HDC hardware literature do not perform the *complete HDC algorithm* on manufactured Application-Specific Integrated Circuits (ASICs) designed specifically for HDC. For example, [72] implements a neural network in software augmented by a key-value Associative Memory in PCM Content-Addressable Memory (CAM) on chip. Ferro-Electric Field Effect Transistors' (FEFET) measured performance characteristics were used to simulate the efficiency of a FEFET-based CAM cell to implement an Associative Memory for HDC in [73]. While [74] has wide applicability due to its instruction-based architecture, its results are simulation-only: it performs simulations on a placed and routed design for 3 applications. However, it is unclear that the routed design is ready for manufacturing as the reported core-area utilization of 70% is extremely high for most technology nodes. The paper [74] does not provide results of manufacturing Design Rule and Electrical Rule checks that can ascertain its manufacturability.

In some papers such as [39, 75], direct measurements on a Field-Programmable Gate Array (FPGA) programmed with the HDC algorithm, instead of a manufactured

ASIC implementing it, are reported. While FPGAs are a great resource for prototyping an application-specific hardware design before manufacturing, its measured energy and performance characteristics often differ from that of ASICs by a factor of $10-50\times$ (for example, see table 3.1.5 discussed in the next chapter).

Most hardware papers study specialized designs *synthesized* into a circuit of logic gates from a standard library. Performance and energy estimates are reported after profiling an application on a compiled program that simulates the synthesized circuit. For example, in [32] a complete and widely-programmable HDC data path was synthesized and profiled by performing simulations of several supervised classification tasks. Similarly, [76, 42, 77] used simulations on synthesized circuits for specialized HDC hardware designs performing supervised classifications on public data sets.

A few papers report measurements of HDC algorithms programmed and run on classical architectures such as CPUs [78, 79] (including on ultra-low-power CPU cores [80, 81, 82]) and general-purpose GPUs [83, 45, 84, 85]. However, as described in section 3.1.5 later on, classical architectures have orders of magnitude higher delay and energy costs than ASICs implementing the same circuit.

A general HDC system capable of supporting multiple applications and HDC algorithms is yet to be *architected, designed, fabricated* and *measured*. Such a general HDC architecture, which can be easily programmed to perform a variety of applications on different datasets, is crucial for evaluating HDC's potential as a viable paradigm for energy-constrained environments [15]. This requires developing the fundamental architectural blocks that are configured and interconnected to produce a complete system. Furthermore, *actual measurements* from a digital CMOS chip in a leading technology node is necessary to establish the anticipated benefits and provide a foundational hardware benchmark for comparing performance benefits of future chips using emerging non-volatile memory such as RRAMs, PCMs, etc.

A comprehensive exploration of such generally-programmable and efficient architectures for HDC is the object of study in this dissertation. The next chapter compares efficiency of HDC on conventional architectures to deduce the likely macroscopic properties of the most suitable architectural family for HDC.

Chapter 3

Principles of constructing an efficient architecture for Hyper-Dimensional Computing

This chapter develops of an *efficient and programmable* architecture for implementing Hyper-Dimensional Computing in custom silicon. The objective is to make the architecture as energy-efficient as possible without sacrificing reasonable programmability. An instrumentation study of HDC and conventional algorithms on embedded CPU and GPU reveals sources of inefficiency when deploying HDC on conventional hardware. Guided by these results and a careful analysis of the structure of HDC algorithms, the prominent macro-properties and most appropriate architectural style are deduced.

The core motif of this chapter is the definition of the *Generic class* of HDC algorithms, and the consequential properties of an efficient architecture supporting it. The idea is that although this class is smaller than the class of all possible HDC algorithms, it contains all known HDC algorithms and is rich enough to likely contain future expert-designed algorithms (discussed further in section 3.3.2). Furthermore, the architecture developed for the Generic class can be easily extended to support more complicated HDC algorithms. After an exposition on the Generic architecture, these extensions are discussed in section 4.2.3 of the next chapter.

3.1 Profiling HDC on embedded CPU and GPU

As noted in section 2.4, a few papers measure the energy and performance of HDC algorithms on CPUs, GPUs and FPGAs. However, no detailed instrumentation on classical architectures, particularly embedded CPU cores (eCPUs) and embedded general-purpose GPUs (eGPUs), has been conducted to date. Instrumenting the execution of a competitive HDC model and comparing with conventional algorithms for the same task will reveal inadequacies of classical architectures for performing efficient HDC. For this comparison, the first step was to decide a set of benchmarks.

3.1.1 Benchmark applications for instrumentation

Since HDC has been widely used for supervised classification [37], 3 popular datasets for supervised classification were chosen for the instrumentation study. Listed in table 3.1.1, these algorithms differ in the amount of available parallelism and belong to different tiers of success by best-known HDC algorithms.

Application	ML algorithm	HDC ($d = 2048$)
EMG Hand-gesture Recognition (EMG)	95.32% (SVM)	95.13 %
Language Recognition (LANG)	97.40% (kNN)	90.82 %
Hand-written digit recognition (MNIST)	99.13% (CNN)	80.28 %

Table 3.1: Inference accuracy of the instrumented algorithms.

1. **EMG hand-gesture recognition (EMG)** classifies electromyography signals sampled from an electrode array attached to the skin of a subject into a set of hand-gestures. The variations in electric and magnetic fields caused due to flexion and relaxation of muscle to produce a gesture changes the electric potential of the electrodes, which are used to recognize hand gestures.

   Support Vector Machines (SVM) are typically used for the EMG by the machine learning (ML) community: more capable algorithms such as Multi-Layer Perceptrons and Convolutional Neural Networks are expensive to train with negligible accuracy gains [86]. A multi-class SVM was profiled for EMG hand-gesture recognition using the dataset of [38].

   EMG has been the most successful application of HDC till date [87, 38]: HDC has higher accuracy and energy efficient than ML algorithms [86] and allows online learning and adaptability to different subjects [88, 39]:

2. **Language recognition (LANG)** was performed on the EUROPARL corpus [1]

   of 21 Indo-European languages transliterated in the Latin alphabet [19] (as explained in section 2.3.2).

   LANG was one of the first applications of HDC to supervised classification and has been widely used as a benchmark in a majority of subsequent work [65].

3. **Hand-written digit recognition (MNIST).** The MNIST dataset of hand-drawn digits is a popular dataset used for evaluating ML algorithms, where 28×28 greyscale pixel-array are used to recognise the drawn digit.

[1] The EUROPARL corpus was extracted from the proceedings of the European Parliament. Due to its diplomatic nature, the sentences are longer and use larger vocabulary than normally used. Thus, EUROPARL is believed to be a difficult dataset for language recognition and translation.

This instrumentation study used a beta release [89], containing 23 languages for training but 21 languages for testing. The 2 additional trained languages Afrikaans and Norwegian are absent from the most recent version [90]. Since Afrikaans is derived from Dutch and Norwegian is similar to Danish and Swedish, their presence as 2 additional trained classes reduces testing accuracy from 93.6% to 90.8% of table 3.1.1. However, since only inference was instrumented, updating the training EUROPARL dataset to the most recent version would negligibly affect results presented here.

MNIST is an elementary dataset in computer vision. It was chosen to represent the fact that known HDC algorithms do not perform well on computer vision applications, especially when using raw pixel values. To evaluate HDC on MNIST, a feature-value superposition (see section 3.3.1) was used on the gray-scale frames [91]. Since the pixel values are bimodal, its values are thresholded first to a 0 or 1 before mapping to 2 random hyper-vectors.

A large number of ML models exist with testing accuracy > 99.5% on MNIST. Usually a committee of a few dozen deep Convolutional Neural Networks (CNNs) are required to achieve such low error rates. However, in this experiment, a single and shallow CNN was profiled for MNIST since instrumenting dozens of state-of-the-art deep CNNs requires access to supercomputing facilities or prohibitively high compute time on commodity processors. The profiled CNN has 2 convolutional layers with 32 kernels of dimensions 3×3 and ReLU activation, followed by 2×2 max-pooling layer and finally a 128-node dense layer (about half a million learned parameters). This CNN is much smaller than the best MNIST classifier but has comparable accuracy (99.13% vs 99.77% for [92]).

Unlike HDC algorithms which need a single pass for training; SVM, kNN and CNNs require multiple iterations – the total number of which depends on initial values of parameters and hyper-parameters. Therefore, only inference accuracy on the test dataset after loading a pre-trained model was profiled.

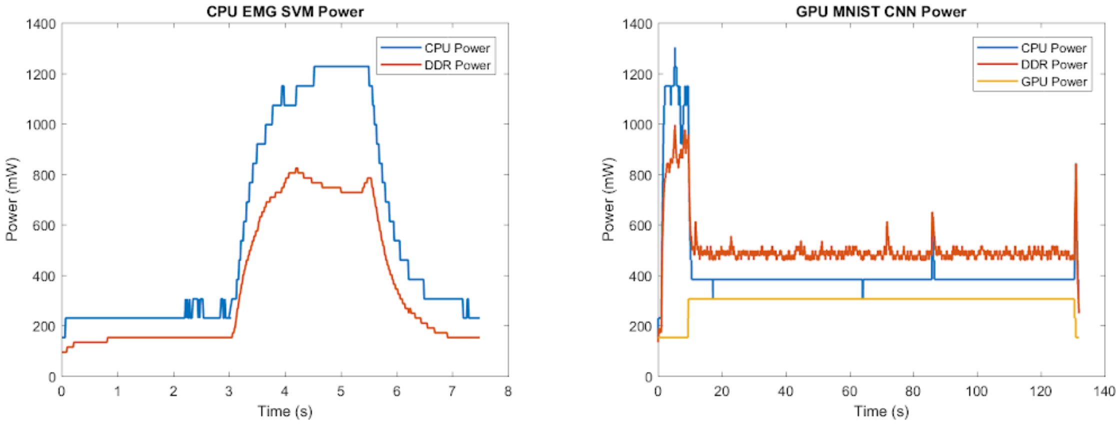
3.1.2 Hardware setup for instrumentation

The NVIDIA Jetson TX2 embedded platform [93] was used to instrument the compiled codes for HDC and its ML counterparts. The Jetson TX2 platform contains a 256-core NVIDIA Pascal eGPU, an ARM Cortex A57 eCPU with 4 cores and 2 megabyte (L2) cache, and 8 gigabytes of 128-bit DRAM (DDR).

Jetson TX2 has an on-board power monitor INA226 [94] used for energy and power measurements. The INA226 power monitor measures power drawn from the separate voltage supplies of CPU, GPU and DDR. This makes it possible to quantify the contribution of each component to the total measured energy/prediction of the profiled algorithm. Power traces are collected from a few seconds before the actual execution of test code begins, so that the start and end time on the traces are clearly visible. Estimates of energy/prediction do not include the initial standby energy. Fig. 3-1 shows an example of power measurement trace for eCPU and eGPU. The eCPU trace (fig. 3-1(a)) illustrates that the power consumption profile is similar to charging and discharging a large capacitor. For the eGPU trace (fig. 3-1(b)), the initial burst of CPU-only activity is followed by a long duration where all 3 components are active – this was observed in all measured traces for eGPU experiments.

3.1.3 Instrumentation results on eCPU

The main goal here to gain a *preliminary understanding* of bottlenecks in classical archictectures for doing HDC, by comparing HDC codebase's instrumentation results



(a) The power trace for running SVM for EMG on ARM Cortex A57.

(b) The power trace for running CNN for MNIST on ARM Cortex A57 & Pascal GPU system.

Figure 3-1: Examples of measured power trace on eCPU and eGPU.

with that of conventional ML algorithms.

Target code: For LANG, HDC and kNN algorithms were written in C [19]. The text-histograms for kNN was generated using a standard C hash function [95]. For EMG, the HDC algorithm [87] was written in C. SVM was implemented in Python using the `LIBSVM` library [96]. For MNIST, both HDC and CNN algorithms were written in C. CNN's C code and the pre-trained model were obtained from [97].

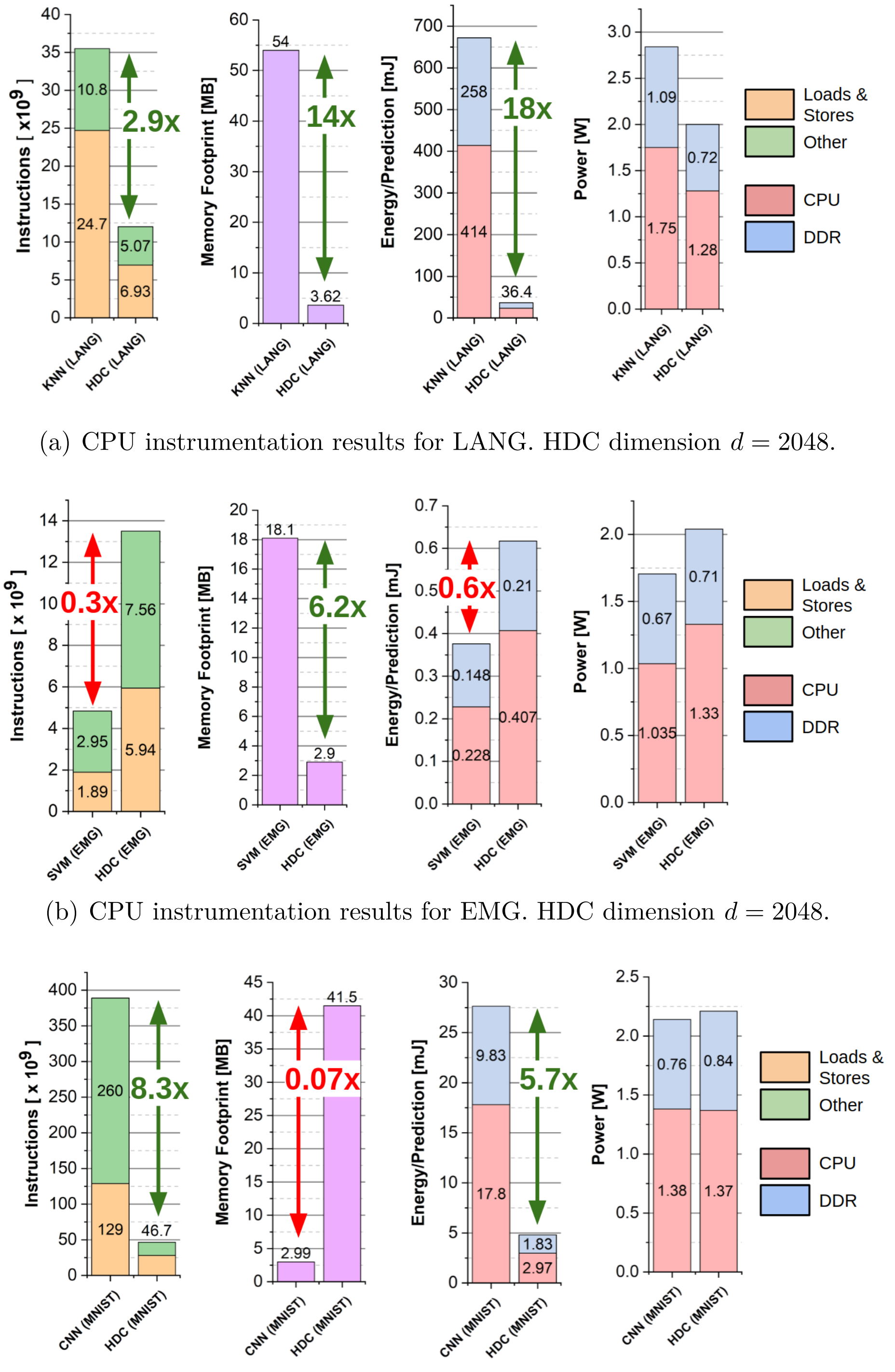
The following quantities were estimated at runtime:

- **CPU instruction count**: the total number of instructions retired by the program during execution. Hardware performance counters were sampled using the `perf-stat` program [98] at about 1000 samples/s. Estimates from 10 independent runs were averaged to reduce empirical variance and the impact of interference by other processes and the operating system.
- **CPU memory footprint**: the maximum page size (including dynamic heap) allocated by the operating system to the program. The `massif` utility of the `Valgrind` framework [99] was used to track pages allocated during execution.
- **Energy/prediction**: average energy cost for each inference by the target code was measured using on-board INA226 power monitor.

All results of CPU profiling are summarized in figure 3-2. The complete data is available in table II of [32]. The ratio of CPU instructions, memory footprint and energy/prediction for the HDC algorithm and ML algorithm is annotated on the figure – a ratio larger than 1 indicates HDC is more efficient.

Some pertinent observations:

- kNN and CNN are inefficient compared to HDC ($d = 2048$), even though hypervectors in C used `int32` to store a bit. kNN is especially inefficient as it compares histograms with $28^4 \approx 5.31 \times 10^5$ dimensions vs $d = 2048$ for HDC.
- SVM is more efficient than HDC ($d = 2048$) on ARM Cortex A57, though it uses a larger DRAM footprint. The lack of bit-level implementation for HDC and



(a) CPU instrumentation results for LANG. HDC dimension $d = 2048$.

(b) CPU instrumentation results for EMG. HDC dimension $d = 2048$.

(c) CPU instrumentation results for MNIST. HDC dimension $d = 2048$.

Figure 3-2: CPU Instrumentation Results.

comparatively smaller dimensions of support-vectors (64 dimensions for SVM vs. 2048 for HDC) may help explain this observation.

- The total power consumption from CPU and DDR voltage supplies for HDC and its ML counterpart are in the same ballpark. In all 3 applications profiled, HDC algorithms spend half to a third of its energy moving data to/from CPU.

3.1.4 Instrumentation results on eGPU

In this experiment, the ARM Cortex A57 CPU (host) and the 256-core NVIDIA Pascal GPU (peripheral device) system on-board Jetson TX was configured to work in tandem: data-parallel kernels (code and data) are transferred from eCPU to eGPU, executed on the eGPU, and results transferred back to the eCPU host.

Target code: HDC code for algorithms used in CPU instrumention in section 3.1.3 were re-written in TensorFlow [100] and Python. This required developing graphs for HDC opearations, creation of data-*tensors* and execution graphs for computing dependant tensors for the final inference accuracy. All HDC tensors use `int32` arrays to store binary hyper-vectors. *k*NN for LANG was implemented in Python/Tensorflow with hash-map dictionaries from the standard distribution. SVM for EMG was implemented in Python/TensorFlow using the `ThunderSVM` library [101]. CNN for MNIST was implemented in Python/TensorFlow and standard CNN libraries.

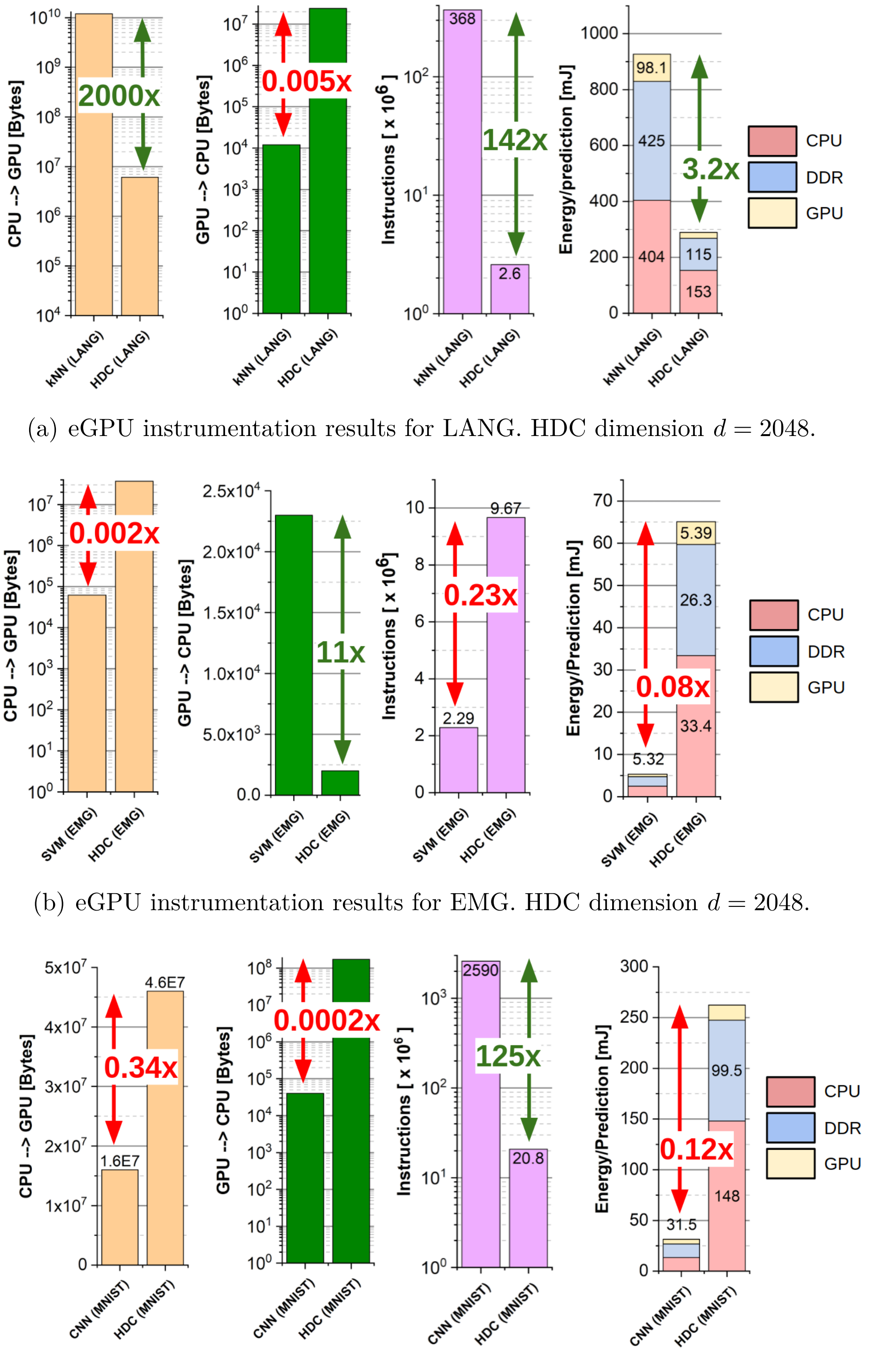
The following quantities were estimated at runtime:

- **GPU instruction count** the number of GPU instructions retired by all kernels in the compiled the program in the execution trace. NVIDIA's proprietary `nvprof` utility was used.
- **Memory (bytes) transferred**: the total memory transfer from host to device and from device to host during execution. NVIDIA's proprietary `nvprof` utility was used: its `API-trace` log contained data used to calculate the total data (in bytes) transferred between eCPU (host) and eGPU (device).
- **Energy/prediction**: average energy cost for each inference by the target code was measured using on-board INA226 power monitor.

Results are summarized in figure 3-3; the complete data is available in tables III and IV of [32]. The ratio of GPU instructions, bytes transferred and energy/prediction for the HDC algorithm and ML algorithm is annotated in the figure – a ratio larger than 1 indicates HDC is more efficient.

Some pertinent observations from this experiment are:

- *k*NN doesn't have enough data-parallelism to efficiently utilize eGPU. It transfers a huge amount of data from eCPU to eGPU before the computation can begin, thereby executing a large number of loads and instructions on the eGPU. Hence, it is highly inefficient.
- As in section 3.1.3, the SVM algorithm for EMG is more efficient than HDC but transfers many times more bytes.
- A downside of using `float32` to store single bits of a hyper-vector in HDC algorithms is the use energy-intensive floating-point ALUs. A hand-optimized GPU kernel code using bit-operations supported by NVIDIA's Pascal architecture could greatly improve HDC performance on eGPU.
- CNN is about 10× more efficient than HDC. This is partially explained by the highly-optimized code for CNN available in TensorFlow's standard library.



(a) eGPU instrumentation results for LANG. HDC dimension $d = 2048$.

(b) eGPU instrumentation results for EMG. HDC dimension $d = 2048$.

(c) eGPU instrumentation results for MNIST. HDC dimension $d = 2048$.

Figure 3-3: Instrumentation Results on the CPU (host) – eGPU (device) system.

However, the fact that far more instructions are executed for CNN than HDC for MNIST indicates its actual computation cost is likely to be small.

Algorithm	Bit-level (BLP)	Ins.-level (ILP)	Thread-level (TLP)	Data-level (DLP)
Hyper-Dim. Computing (HDC)	High	Medium	Low	High
k-Nearest Neighbors (kNN)	Low	Low	Low	Medium
Support Vector Machines (SVM)	Medium	Medium	Medium	Medium
Conv. Neural Nets. (CNN)	High	Medium	Medium	High

Table 3.2: Levels of parallelism present in instrumented algorithms.

3.1.5 Lessons learned

To organize results from sections 3.1.3 and 3.1.4, it is helpful to characterize the types of parallelism present in HDC, kNN, SVM and CNN. An algorithm may exhibit 4 classical levels of parallelism [102]: bit-level parallelism (BLP) exploits full-width logic units for complex arithmetic, instruction-level parallelism (ILP) exploits concurrent execution of logically-sequential instructions without architectural hazards, thread-level parallelism (TLP) exploits concurrent execution of independent processes, and Data-level parallelism (DLP) performs identical operation on different data points simultaneously. An efficient hardware exploits most levels of parallelism present in the workload. Table 3.2 lists the levels of parallelism available in the profiled algorithms.

kNN has low parallelism because generating sub-sequences and maintaining a large histogram in memory is a great bottleneck. Thus, it performs poorly for both eCPU and eGPU. SVM has moderate ILP and TLP exploited by eCPU, and some DLP to make efficient use of eGPU. Its low ILP and very high DLP indicates the eGPU for CNN is a better choice.

HDC performs simple logical operations on high-dimensional vectors, hence it has very high BLP. However, the ALUs in eCPU and eGPU unable to harness it using bit-operations on parts of a hyper-vector (unless specialized assembly code is implemented). The feature-value superposition algorithm also allows multiple channels to be processed concurrently (i.e. having high DLP). Unfortunately, compilers cannot discover the underlying parallelism present in HDC as it is hard to discover the coherence of very-large dimensional hyper-vectors (d is in thousands) with an instruction set operating on 32 or 64 bits. HDC still produces more efficient code than ML occasionally.

Finally, a comparison across platforms in figure 3-4, reveals that an ASIC implementation can offer *orders of magnitude* improvement in classification energy, with energy/inference of a few micro-joules necessary for deployment in sensor-based IoT.

Hand-optimized FPGA code [103] also results in order(s) of magnitude higher energy/prediction cost than ASIC (see table 3.1.5; ASIC results for HDC $d = 10^4$ from section 4.3.2 and section VII of [32]). HDC ($d = 10^4$) for EMG was profiled for inference energy by [82] on the PULP SoC containing ultra-low-power CPU cores. The PULP SoC uses advanced circuit techniques such as near-threshold operation, optimized implementation of bit-wise extensions of OpenRISC ISA and voltage supply reduction [104]. A hand-written code optimized for exploiting TLP available in HDC was deployed on the 4-core cluster, however the energy/inference costs are $> 5\times$ that of ASIC implementation in the same technology node: 42 μJ/prediction for single-core execution on PULP and 21 μJ/prediction for 4-core execution on PULP vs ≈ 4 μJ/prediction on synthesized ASIC implementation (section 4.3.2).

To summarize, the case for developing accelerator for HDC is an exemplar for the *accelerator-level parallelism* concept proposed recently [102]: HDC uses far higher data-widths and very different operations on its data than instructions for classical architectures. Thus, the next step is to develop a simple architecture for an HDC accelerator, composed of simple building blocks and a small programmed state.

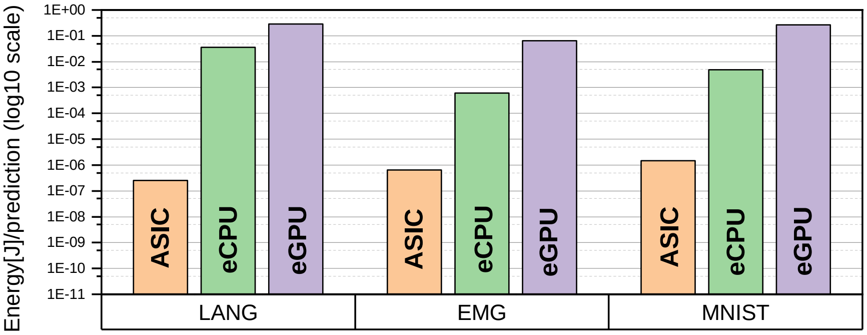


Figure 3-4: Energy/prediction summary across platforms.

HDC ($d = 10^4$) platform	Dataset: UCIHAR	Dataset: ISOLET	Dataset: FACE
Hand-optimized FPGA code	$1 - 10$ mJ	$10 - 100$ mJ	$1 - 10$ mJ
Post-synthesis 28nm ASIC	< 22.1 mJ	< 11.6 mJ	< 18.7 mJ
FPGA/ASIC energy	$> 45\times$	$> 860\times$	$> 54\times$

Table 3.3: Comparison of ASIC implementation of HDC with hand-optimized FPGA implementation [103] for dimension $d \approx 10000$.

3.2 Structure of HDC algorithms

The goal of this section is to deduce the likely properties of an efficient HDC architecture from the properties of MAP operations (from section 2.2). To achieve high energy efficiency, the unifying principles of simplicity, optimal use of resources and avoiding redundant computation are employed.

A successful and widely-applicable architecture must be able to use application-specific data from a variety of domains after suitable pre-processing. The first step towards such a general design is to abstract essential elements of *all possible* HDC algorithms using the MAP framework. As shown below, a clear structure emerges.

3.2.1 Value representation in HDC

To allow consumption by a discrete-time (clocked) digital system, the input data must be quantized into discrete states and sampled with a finite frequency. The choice of quantization scheme and sampling rates are important [105, 106] and is assumed to be pre-determined by a domain expert. Hence, without losing generality, any HDC algorithm's input space can be abstracted as a symbol set $\mathbb{X}'$ representing values in the domain's feature space. For example, MNIST from sec 3.1.1 has symbols for each pixel: `pixel1`, `pixel2`, ... `pixel784` and for each pixel value: $0, 1, \ldots, 255$. To handle multi-channel inputs, multiple streams can be serialized to a single stream with a suitable policy (for example, samples from 2 channels $x_1, x_2, ...$ and $y_1, y_2, ...$ combined to $x_1, y_1, x_2, y_2, ...$). The order of serialization depends on data acquisition order and buffering capacity; for streaming applications only minor shuffling on raw data is usually feasible at runtime.

Therefore, the input is abstracted in an application-agnostic manner as a single finite-length time-series of symbols and denoted $I_{serial} \triangleq (x_1, x_2, ..., x_T)$. For example, an MNIST image may be represented by the input stream $I_{serial} =$ (`pixel1`, `<value of pixel 1>`, `pixel2`, `<value of pixel 2>`, ..., `pixel784`, `<value of pixel 784>`). Note that I_{serial} is the input stream after sampling, all domain-specific pre-processing, quantization, and channel ordering are completed.

3.2.2 Encoding stages

An input symbol is substituted by a hyper-vector (defined as **item**) to allow computations using them with the MAP operations. If the symbol doesn't represent entities from an ordered set (such as `pixel1`, ... `pixel784` for MNIST) they are assigned to random hyper-vectors. If it represent numbers or vectors from an ordered field, some transformations on random hyper-vector is performed before assigning it to the symbol [87]. Hence, the symbol set $\mathbb{X}'$ representing the input space is substituted by an item set $\mathbb{X}$. Let $\mathbb{N}_n \triangleq \{1, 2, ...n\}$ be the set of all natural numbers up to n. Therefore, the input stream of symbols $I_{serial} = (x_t | t = 1, 2, ..., T)$ (where each $x_t \in \mathbb{X}'$) is assigned to a stream of items $\mathbb{I} \triangleq (X_t | t = 1, 2...., T)$ (where each $X_t \in \mathbb{X}$). Any collection of input symbols can be specified by set of indices in $\mathbb{I}$.

To discern the basic properties of HDC algorithms, it is important to consider its inherently symbolic nature. Each input symbol is substituted by an item that are used to compute *expressions* using HDC operations (multiply, permute and superposition for MAP) to form hyper-vector representations of composite structures (such as list, trees, classes for supervised classification). Historically, HDC's remarkable power for analogical and hierarchical reasoning exploiting such composite representations was the first to be discovered [63, 27].

Any HDC algorithm is composed of multiple steps – each step is either processing (i.e. "encoding") the stream of items substituting the stream of input symbols using the MAP operations to produce a composite hyper-vector, or comparing a computed composite hyper-vector with all previously stored hyper-vectors in an associative data-structure to return the closest match. While the number and order of steps and the details of encoding items vary for different HDC algorithms. the comparison of vectors in an associative data-structure is an identical operation. Therefore, the study of HDC algorithms can be reduced to study of encodings and of the combination of different encodings and the associative search.

An encoding step is an expression of hyper-vectors $X_t, t \in \mathbb{N}_T$ and MAP operations multiply $\oplus$, permute $\rho()$ and addition i.e. (thresholded-)superposition $[\ldots + \ldots]$. Importantly, using these laws of the MAP operations, any encoding expression can be transformed to have superposition as the last operation:

- $\rho([A + B + C + \ldots]) = [\rho(A) + \rho(B) + \rho(C) + \ldots]$ for all permutations $\rho()$ and vectors $A, B, C, \ldots$
- $Z \oplus [A + B + C + \ldots] = [(Z \oplus A) + (C \oplus B) + (Z \oplus C) + \ldots]$ for all vectors $A, B, C, \ldots, Z$ since the threshold after superposition is the element-wise mean (0 for $+1/-1$ vectors). $[A+B+C+\ldots]$ computes the bit-wise majority function, and element-wise XOR ('$\oplus$' in the MAP model) distributes over it.
- $A = [A]$ for all vectors A.

Without loss of generality, all HDC encoding expressions are assumed to have superposition as the last operation. This allows the classification of encoding expressions by number of nested hierarchies of superposition present in them.

A **single-stage** encoding expression contains superposition only once, as the final operation. The expression is $S = [\sum_{i=1}^{K} f_i(\mathbb{I})]$ where i^{th} term hyper-vector is:

$$f_i(\mathbb{I}) = (X_{p_1} \oplus X_{p_2} \ldots \oplus X_{p_m}) \oplus (\rho^{u_1}(X_{q_1}) \oplus \rho^{u_2}(X_{q_2}) \ldots \oplus \rho^{u_n}(X_{q_n}))$$

where each term $f_i(\mathbb{I})$ contains specific items and their permutations: some items as is (set of positions in $\mathbb{I}$ denoted by $P_i \triangleq \{p_1, p_2, \ldots, p_m\} \subseteq \mathbb{N}_T$), and others are *permuted* (set of positions in $\mathbb{I}$ denoted by $Q_i \triangleq \{q_1, q_2, \ldots, q_n\} \subseteq \mathbb{N}_T$, the permutation degrees $u_1, u_2, \ldots, u_n$ are positive integers). Note that a few inputs may occur both with and without permutation in the term (i.e. $P_i \cap Q_i \neq \phi$).

A **dual-stage** encoding has the final superposition with terms composed of products of items and of single-stage expressions only, and their permutations.

Similarly, all multi-stage encoding expressions are defined.

3.3 The Generic architectural model for HDC

Following arguments from the previous section, the complexity of supporting all possible encodings can be hierarchically reduced to the complexity of implementing single-stage encodings. However, an architecture attempting to support all possible single-stage encodings is very likely to be highly inefficient for most encodings (like eCPUs/eGPUs from sections 3.1.3 and 3.1.4), as it must support all possible input dependency patterns. The number of all possible single-stage encodings is *extremely large*, as can be seen by a simple counting argument. Since permutation has a period of d (from section **??**), each term in a single-stage encoding expression is a product of distinct, non-empty selections from the set $\{X_1, X_2, \ldots, X_T, \rho(X_1), \rho(X_2), \ldots \rho(X_t), \ldots, \rho^{d-1}(X_1), \ldots, \rho^{d-1}(X_T)\}$ containing dT distinct elements. Thus there are $2^{dT} - 1$ distinct possibilities for each term. If all terms of the superposition were distinct, the total number of possible single-stage encodings are $2^{2^{dT}-1} - 1$. Hence there are $\Omega(2^{2^{dT}})$ possible single-stage encodings, a quantity *super-exponential* in HDC dimension d (usually in thousands) and number of symbols T (usually in hundreds). More rigorously, observe that every term-expression may be represented by dT bits: for each element of $\{X_1, X_2, \ldots, \rho(X_1), \ldots, \rho^{d-1}(X_1), \ldots, \rho^{d-1}(X_T)\}$, a 1 indicates its presence in the term and 0 indicates absence. Hence, every single-stage encoding expression has a corresponding dT-bit boolean function, whose output is 1 iff the term represented by the dT-bit input is present in the expression's superposition. A classical result from circuit complexity (Theorem 7 of [107]) establishes that this requires at least $\Theta(2^{dT-\log dT})$ logic gates – a number almost exponential in HDC dimension d and number of symbols T. Clearly, to support all (single-stage) encodings *and* guarantee overall efficiency is not feasible.

Therefore, a prudent approach is to support a limited collection of encodings very efficiently – one which contains all successful and useful HDC algorithms known till date. This strategy is likely to succeed since most HDC algorithms were designed by experts rather than being produced by an automated procedure (such as deep learning), and given that HDC derives most of its efficiency from short programs/encodings [19, 86], it is likely to use similar expressions repeatedly.

3.3.1 Common algorithmic kernels

Remarkably, variations of only 2 encoding expressions are used in a great majority of known HDC algorithms:

- **n-gram sequence encoding**. As mentioned in Section 3.1.1, n-grams i.e. the multi-set of n-sequences can be very useful for modelling sequences. Characters in an symbol set $\mathbb{A} \triangleq \{a_1, a_2, ..., a_N\}$ are mapped to random hyper-vectors $Y_{a_1}, Y_{a_2}, ..., Y_{a_N}$ and the n-sequence $x_1, x_2, ...x_n$ (where $x_i \in \mathbb{A}$) is encoded as $\rho^{n-1}(Y_{x_1}) \oplus \rho^{n-2}(Y_{x_2}) \oplus \rho^{n-3}(Y_{x_3}) ... \oplus Y_{x_n}$. Finally, all occurring n-sequences in the input are encoded and superimposed to form the final hyper-vector.
- **Feature-value encoding**. Let the feature vector of f dimensions be $V = (v_1, v_2, ..., v_f)$. For each feature $i \in \mathbb{N}_k$, a random hyper-vector C_i is assigned and as discussed in section 3.2, all possible input values v are assigned items Y_v.

V is encoded as $[\sum_{i=1}^{f}(C_i \oplus Y_{v_i})]$ and a collection of n samples (a matrix U with sample vectors as rows) is encoded as the superposition $[\sum_{i=1}^{n}\sum_{j=1}^{f}(C_j \oplus Y_{U_{ij}})]$.
From section 3.1.1: superposition of 4-grams is used for LANG, EMG uses a 2-stage encoding: 64-feature as the first stage and 4-grams as the second stage. HDC for MNIST encodes superposition of 784-feature samples.

3.3.2 The Generic abstraction

The main complexity in encoding is the generation of term vectors $f_i(\mathbb{I})$. Limiting the encoding to only n-gram sequence and feature-value encodings make these term expressions fixed. The total number of possible single-stage encodings is now only $\Theta(dT^2)$ since number of feature-encodings is $dT(2d-1)$, and number of n-grams is T. Therefore, only n-grams and feature-value encodings are too restrictive.

The proposed set of encodings should generalize the *regularity* present in n-gram and feature-value encodings. For instance, note that each term of these two encoding schemes require only few specific inputs $A_i = P_i \cup Q_i \subseteq \mathbb{N}_T$ (defined the set of **dependent inputs**), which a small part of the total number of symbols T.

Therefore, the following conditions are proposed:

1. **Constant number of inputs per term.** The number of dependent inputs $|A_i|$ is constant for all terms $i \in \mathbb{N}_K$. (Call this quantity by L.)
2. **Common expression for all terms.** All K terms have the *same HDC expression.* If $f_i(z_1, z_2, ..., z_L)$ denotes the expression of the i^{th} term in terms of L *input variables* $z = (z_1, z_2, ..., z_L) \in \mathbb{X}^L$, then $f_i(z) \equiv f_j(z)\ \forall i, j \in \mathbb{N}_K$. This **common term expression** will be called $f(z)$.
3. **Term inputs are continuous chunks of the input streams.** The set of dependent inputs for the i^{th} term are translations of a fixed sub-sequence of indices in the input stream. That is, for some increasing sequence $t_i \in \mathbb{N}_T$ with $t_1 = 0$; the i^{th} input dependency set is $A_i = 1 + (A_1 + t_i) \mod T$ where $A_1 + t$ denotes the set obtained by adding t to each element of A_1.

These conditions completely specify the collection of all (single-stage) encoding expressions supported by the architecture studied in this dissertation, and shall be called **Generic encodings**. An architecture designed to support all Generic encodings shall be called a **Generic architecture** (as opposed to “general-purpose”, “complete”, “fully programmable” or “general”).

Observe that the regularity conditions above clarify important structures of a Generic architecture. Since inputs to each term is a translation of the overall input time-series, the entire programming complexity of such a machine is only for computing $f(z)$. Property 3 above ensures an in-order sequential generation of all K term vectors by any pipelined hardware for $f(z)$ with fixed latency and throughput equalling the symbol input rate. As the input symbols $\mathbb{I}$ are entered in sequence, the i^{th} term is produced t_i steps after the first term. The *final superposition* $S = [\sum_{i=1}^{K} f_i(\mathbb{I})]$ of a single-stage Generic encoding may be computed by accumulating these terms $f_i(\mathbb{I})$ at the required time-steps $(t_1, t_2, ...)$. A T-bit register can enable these clock cycles to allow the accumulation of the generated term hyper-vectors.

Finally, note that the class of Generic algorithms is rich and large: there are $2^{dL}-1$ term expressions possible for each $1 \leq L \leq T$, and T translations (of length L) possible for each term. Therefore, assuming unique terms being superimposed, there are $(2^{dL}-1)(2^{T}-1)$ unique single-stage Generic encodings. Thus, the total number of Generic encodings possible with T input symbols and HDC dimension d is $\sum_{L=1}^{T} \Omega(2^{T+dL}) = \Omega(2^{(d+1)T})$.

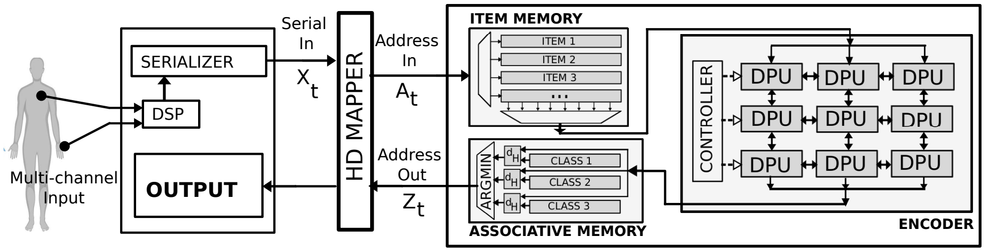


Figure 3-5: The major components of a HDC processor.

3.3.3 Major components of the Generic architecture

A generic HDC architecture requires 3 fundamental components:

- **Item Memory (IM)** maintains a repertoire of random hyper-vectors (items). A sufficiently large collection of such vectors can be re-used for many applications. This storage requirement cannot be avoided as the map of items assigned to input symbols must be retained for the entire life of the application.
- **Encoder** combines its input hyper-vector sequence using the MAP operations to compute a Generic expression specific to the application.
- **Associative Memory (AM)** stores all encoded hyper-vectors representing composite concepts and associatively searches against a query hyper-vector to return the closest-match among them.

  For supervised classification, the Associative Memory stores an encoded vector as a representation of a class after training, and returns the closest match to an encoded test query as the prediction while testing.

Fig. 3-5 shows a diagram for the complete Generic HDC system, illustrated for a human-centric IoT application. All application specific pre-processing, sampling and quantization is done before combining input streams. The peripheral **HD Mapper** block assigns incoming symbols to an address in Item Memory and class labels to an address in Associative Memory. This mapping is retained by HD Mapper throughout all sessions of the live application. Therefore, the actual inputs and outputs to the application-agnostic Generic processor is a time series of addresses of memory.

Fig. 3-5 also shows the operation during testing when applied to supervised classification. The HD Mapper substitutes the input symbols for Item Memory addresses. The Item Memory fetches item hyper-vectors and passes them on to the encoder, which generates the test hyper-vector according to the encoding expression programmed by the user. The Associative Memory returns the address of the closest class vector as output, which is converted by HD Mapper to the predicted class label.

3.3.4 Arguments for a data-flow architecture

While there could be multiple ways to interconnect these fundamental blocks, the simplest system obeys:

1. **Single programmable component**. Only the Encoder needs to be programmed for an application. The operation of both memories always remain the same and do not change with applications. The simplest system will have a single Item Memory, Encoder and Associative Memory; the sole Encoder is the only programmable component.
2. **Uni-directional data-flow**. In the *simplest* case, hyper-vectors move from Item Memory to Associative Memory through the Encoder. This is obeyed by all single-pass HDC algorithms for supervised classification. As shown in section 4.2.3, an architecture supporting uni-directional data-flow can be easily modified to support newer HDC algorithms such as factorization [108] requiring multiple iterations of hyper-vectors moving between Encoder and Associative Memory in a cycle.

Note that since Encoder is the only non-memory and the sole programmable component, all major architectural decisions principally concern the Encoder. Furthermore, note that all MAP operations performed by the Encoder are simple and bit-parallel – superposition and multiply are element-wise operations. And although permutation creates across-vector dependency, the pattern is very *regular* and connects to a fixed neighbouring element. These observations can help guide the selection of the architectural style for the Encoder.

To begin with, the generic model of section 3.3.2 explicitly encodes dependencies and bit-level parallelism not exploited well by compiled code using classical instructions and architectures. Hardware required to extract dynamic ILP and DLP add to energy costs. Section 3.1.3 and 3.1.4 and fig. 3-4 show that conventional architectures such as embedded CPU and embedded GPUs are very energy-inefficient for HDC. Vector and other Single-Instruction-Multiple-Data (SIMD) architectures with full data-width d are expensive. HDC algorithms are too small to extract significant run-time parallelism to justify their overhead. Finally, such architectures require wide register-files and complex control logic requiring extra overhead and contradicting the advantages of performing a few steps composed of simple bit-wise MAP operations. Considering these factors, a *data-flow based array architecture* is the most suitable style.

Here, the Encoder is comprised of a regular network of simple Data Processing Units (DPU), and inter-DPU communication for dependencies is restricted to neighbors no more than a fixed distance away [109]. Though several attempts have been made to map common workloads to DPUs [110, 111, 112, 113, 114], only a few of them where dependency patterns can be expressed as a regular graph have been successful [109, 115, 116]. HDC algorithms perfectly fit these conditions. All MAP operations can be implemented with a few gates. The sequential symbolic-input model of section 3.2.1 and the Generic abstraction of section 3.3.2 enables one to map any HDC algorithm to DPUs explicitly. The next chapter develops these details for the Encoder and the Generic architecture.

Chapter 4

Programmability, scalability and a hardware evaluation of the Generic HDC architecture

This chapter considers in detail the micro-architectural choices of a generic HDC architecture implementable in Silicon. Following from the definition of the Generic HDC architecture the first section examines the building blocks and construction for the Encoder – the only programmable component. Descriptions of its programmability, methods of controlling its execution, pipeline states, sparsity-based energy-efficiency and security considerations are provided. This completely specifies the core features and capabilities of the Generic architecture studied in this dissertation. Next, the three major components of the generic architecture (i.e. Item Memory, Encoder and Associative Memory), their extensions and interconnections to produce a generic HDC system of the required capability is considered. Finally, an evaluation of a generic HDC system, synthesized in an advanced technology node, is performed on a benchmark of supervised classification tasks.

4.1 Organization of the Encoder

The Encoder is crucial for the overall programmability of the processor and has the largest activity and wiring complexity. Hence, its efficient and minimal design is important for an optimal implementation of the architecture. As outlined in section 3.3.4, encoders must be organized into multiple redundant computing elements as a consequence of adopting the data-flow architecture style.

The Generic architecture (section 3.3.2) is based on the fact that computing any expression of the MAP model for a single-stage algorithm can be decomposed into:

- the generation of all the necessary terms $f_i(\mathbb{I})$ from each of the dependent input symbols entered into the processor, and
- computing their superposition to evaluate the result $S = [\sum_{i=1}^{K} f_i(\mathbb{I})]$.

Therefore, the first principle of organization is to split the stage of Encoder into two functional parts: one for collecting all dependent inputs and generating the terms (*producer of terms*), and the other for collecting the terms as they are produced, scheduling them for accumulation, and performing the binary thresholding operation once the stage's input symbols are exhausted (*consumer of terms* or *accumulator*). Since the accumulator is a common resource shared by all terms in a stage of encoding algorithm, the main decisions about its construction are about scheduling terms for accumulation, avoiding conflicts (i.e. terms produced in the same cycle must be stored for accumulation in separate cycles) and properly initiating the thresholding operation. In contrast, different terms of generic expressions may not share many input symbols. But the conditions of the generic model ensure the expressions have a regular structure: they are all the same term expression $f(\cdot)$ but different operand symbols. The common term expression over different inputs strongly suggests that a network of DPUs (proposed in section 3.3.4) be employed for the production of terms, and hence it needs to implement only multiply and permute. The accumulator implements the superposition/add HDC operation.

Note that the encoding and fetching of hyper-vectors from Item Memory should occur in lock-step. The associative memory cannot begin until the entire vector has been encoded. Hence, a computation phase barrier exists between the Encoder and Associative Memory. The encoder DPUs can be designed to process narrow slices of HDC vectors and the network to join them together. Since HDC operations are very simple, such DPUs should be small and easily optimized. Locality of operand dependencies ensures that the network is regular and free from wire congestion. Finally, as HD algorithms are short and online [14], the entire network could be small and have a short depth. This reduces the distance between Item memory (the item store), Encoder (computations) and Associative memory (the vector store), thereby substantiating the expectation of HDC being an example of in-memory computing.

4.1.1 Hyper-dimensional Logic Unit (HLU)

Since the network of DPUs need to perform only multiply/XOR and permutation in the MAP framework of HDC, the details of the DPU abstraction and their interconnections follow as a logical consequence.

To begin with, observe that the simplest possible DPU is that which supports single-bit multiply/XOR and permutation: fig. 4-1(a) shows the **Hyper-dimensional Logic Unit (HLU)** containing a single-bit flip-flop and simple gates. It takes two input bits A and B every cycle, a permutation input bit `p_in`, and can "multiply" (`out`$= A \oplus B$), "permute" (`out` $= \rho(B) =$ `p_out`), "delay A by a cycle" (`out` $= A$) or "permute and multiply" (`out` $= A \oplus \rho(B)$). The permutation output bit `p_out`, the output bit when permutation is involved, is just the input bit B.

Since permutation produces intra-word dependencies in a hyper-vector, d identical HLUs can be connected together by chaining inputs `p_in` and `p_out` of different HLUs, according to the permutation implemented, to form a module operating on an entire hyper-vector (Fig. 4-1(b)). This coherent unit will be called **HLU Layer**. It takes two hyper-vector operands A and B every cycle, produces hyper-vector `OUT` and can "multiply" (`OUT`$= A \oplus B$), "permute" (`OUT`$= \rho(B)$), "delay A by a cycle" (`OUT`$= A$) or "permute and multiply" (`OUT`$= A \oplus \rho(B)$). Each constituent HLU performs the *same*

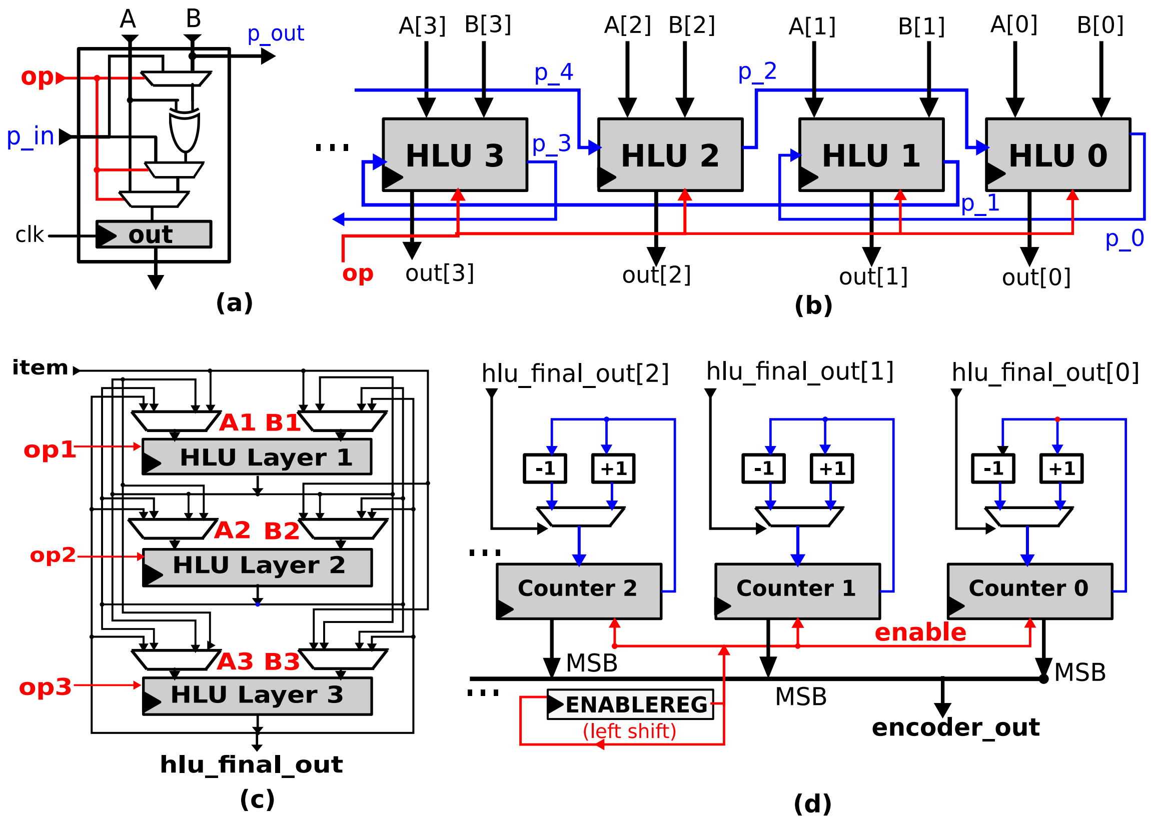


Figure 4-1: Encoder organization.
(Clock-wise from top-left): (a) Hyper-Dimensional Logic Unit (HLU), (b) HLU Layer: connecting single-bit HLUs to create a HLU Layer for multiplying and permuting hyper-vectors, (c) HLU Layer Network: inter-connecting multiple HLU Layers to generate terms, and (d) Accumulator for superposition and thresholding of terms. Control and data-path wires have distinct colors.

operation on their respective bits of the operand hyper-vectors. The actual operation performed is determined by the 2-bit `op` signal provided by the Encoder's control logic. The permutation must be a single-cycle derangement, hence any hamiltonian path connection through `p_in` and `p_out` visiting all HLUs is valid. Fig. 4-1(b) illustrates a scheme where alternate HLUs (except first and last) are connected to minimize length of longest wire for a linear physical placement constraint on the HLU Layer.

HLU Layers can be interconnected among themselves to form a **HLU Layer Network**, generating an overall output `hlu_final_out` every cycle by operating on the stream of input hyper-vectors consumed thus far (labeled `item` in fig. 4-1(c), usually item hyper-vectors directly from the Item Memory). Observe that each HLU Layer is a pipeline stage storing intermediate hyper-vectors as they contain d-bit flip-flops. By convention, the last HLU Layer's output will be considered `hlu_final_out`.

Finally, the **Accumulator** is a d-dimensional array of counters for performing superposition and thresholding of outputs from the HLU Layer Network (see fig. 4-1(d)). A counter at each hyper-vector position stores a signed integer in two's complement representation, and increments or decrements it every clock cycle according to corresponding bits of `hlu_final_out` being 1 or 0 respectively. When the encoding completes, the vector formed by the MSB of the counters is the required superposition after thresholded. An update by increment/decrement at *all* counters can be suppressed or enabled each cycle using the output of an always circular-left-shifting register `ENABLEREG` (see fig. 4-1(d); let 0 represent suppression). `ENABLEREG` is pre-loaded with an appropriate bit pattern so that its contents are shifted as the design encodes, enabling accumulation of *only the required terms* for the HDC expression.

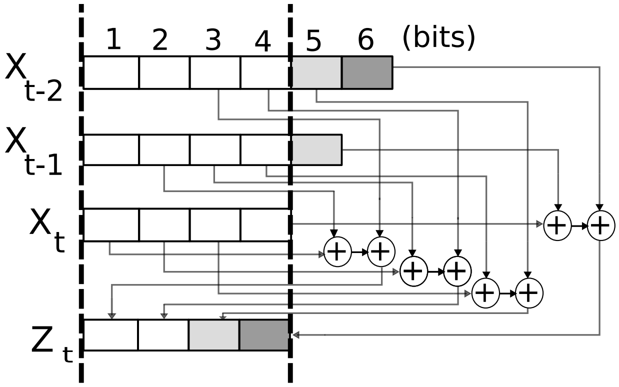


Figure 4-2: Permutation leads to across-word dependency in the Encoder. The across-word dependencies created by permutation when evaluating a 3-gram Z_t of hyper-vectors X_t, X_{t-1}, X_{t-2} is shown; the $\oplus$ symbol *in this figure* represents element-wise accumulation before thresholding to produce Z_t.

Now that the main structure of the HLU Layer Network is architected, the width of the network must be decided. The question to be resolved is: is the full-width HLU Network of HDC dimension d optimal, or is a smaller word more efficient?

To begin answering, first note that 2 of the 3 fundamental components, the Item and Associative Memories, must be of full-width d to avoid accesses to extremely expensive off-chip memory. Secondly, since the critical path is determined by the most complex logic, likely to be the Accumulator of the Encoder or the distance calculation block of the Associative Memory – both independent on HDC dimension d), reducing the HLU Network's dimension is unlikely to change the clock period significantly. Thus, the following arguments suggest width $< d$ is likely to be inefficient:

- **Sub-word HLU Network requires iterations**. Encoder words of width $< d$ introduce multiple iterations on any HDC algorithm – especially single-pass algorithms. This necessitates storing data which can have a large memory footprint. For example, millions of `char` in EUROPARL from section 3.1.1 would require mega-bytes of storage. For a full-width HLU Network, this is not necessary – especially for commonly occuring single-pass HDC algorithms.
- **Permutation leads to across-word dependency**. Illustrated in fig. 4-2: at each cycle, grey-coded output bits require operands from neighboring sub-words due to permutation. This leads to redundant computation as the extra bits from

a neighbouring sub-word. For example: n-gram uses $n(n-1)/2$ extra bits/cycle for each sub-word.

Therefore, the HLU Network is recommended to be of full-width d where the HDC dimension d must be large enough to support a majority of HDC applications.

4.1.2 Programming the Encoder

An important decision regarding the design of array architectures is the choice of interconnection network for the DPUs. A general interconnection network allows efficient mapping of a largest possible collection of algorithms at the cost of increased programming state, hardware complexity and power consumption. For the HLU Layers in a HLU Network, the most general interconnection network allows *feedback* i.e. it allows connecting the output of any HLU Layer or the Encoder's input (the item for unidirectional HDC processors) to either one of the two hyper-vector inputs A and B in each HLU Layer. In particular, a HLU Layer's input may be programmed to be its own output from the previous cycle. Such a **fully-connected HLU Network** with 3 HLU Layers is shown in fig. 4-1(c). Signals `op1, op2, op3` program the operation carried out by HLU Layer $1, 2, 3$ respectively. Since the term expression to be implemented by HLU Layers is constant for Generic expressions, these control signals stay constant throughout encoding. Operand-select signals `A1, B1, A2, B2, A3, B3` decide the actual interconnections of HLU Layers.

HLU Layers are the principal computing resource in this encoder architecture. They are meant to hold intermediate results, a much more efficient strategy than using expensive high-dimensional register-files or a data cache. A possible simplification to the fully-connected network is to allow feed-forward connections only i.e. no cycles in input-output dependency paths across HLU Layers are allowed. For example, figure 4-3(a) shows a feed-forward interconnection and programming of 3 HLU Layers to encode the 3-gram. Fig. 4-3(c) shows an alternative interconnection using feedback for HLU Layer 1 to produce 3-grams with 5 HLU Layers.

Using feedback allows multiple configurations for encoding a term expression and often produces interesting intermediate expressions. For example, the first layer of the HLU Network in figure 4-3(c) encodes an infinite-length term expression: $Z_t \triangleq X_{t-1} \oplus \rho(X_{t-2}) \oplus \rho^2(X_{t-3}) \oplus \rho^3(X_{t-4}) \oplus ...$ (as mentioned in fig. 4-3(b)). While using this to produce 3-gram requires 5 HLU Layers instead of 3, it may have significant advantages from a security point-of-view (see section 4.1.4). The remaining steps to produce a 3-gram from Z_t can be understood as follows: observe that $A \oplus A = 0$ and $A \oplus 0 = A$ for all vectors A. The output Z_t of HLU Layer 1 can be re-written as $Z_t = X_{t-1} \oplus \rho(X_{t-2}) \oplus \rho^2(X_{t-3}) \oplus \rho^3(Z_{t-3})$. HLU Layers $2, 3, 4$ permute and add a cycle delay to Z_t to produce $Z'_t \triangleq \rho^3(Z_{t-3})$ as the output of HLU Layer 4. Finally, HLU Layer 5 computes $Z_{t-1} \oplus Z'_{t-1} = Z_{t-1} \oplus \rho^3(Z_{t-4}) = X_{t-2} \oplus \rho(X_{t-3}) \oplus \rho^2(X_{t-4}) \oplus \left(\rho^3(Z_{t-4}) \oplus \rho^3(Z_{t-4})\right) = X_{t-2} \oplus \rho(X_{t-3}) \oplus \rho^2(X_{t-4})$ which is the term expression required for encoding the 3-gram.

The total number of wires, which determines physical routing complexity, grows quadratically with the number of HLU Layers in the fully-connected interconnection network. Hence, encoders with a large number of HLU Layers are unlikely to allow

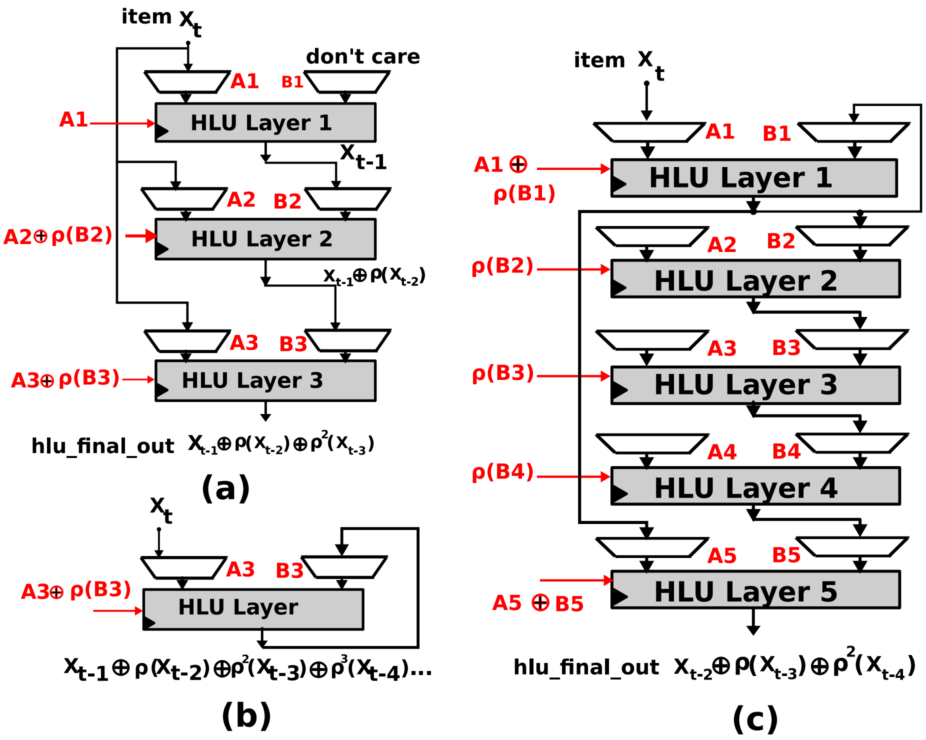


Figure 4-3: Examples of programming 3-gram in a HLU Network. Control signals of HLU operation and operand-select are colored red. An unused operand of a HLU Layer is left unconnected or marked "don't care". (a) An example of feed-forward-only connections for programming 3-gram. (b) Using feedback allows construction of a rich variety (including infinite length) term expressions. (c) An alternative configuration of HLU Layers using feedback for 3-gram.

feedback. As discussed in the next section, feedback also complicates the pipeline control logic as flushing and filling a pipeline with arbitrary feedback is difficult.

Since n-grams require at least n pipeline stages to store each of the hyper-vectors for the n-subsequence *with or without feedback*, the feed-forward implementation of figure 4-3(a) (shown for $n = 3$) is optimal. Note that more than the minimum number of HLU Layers is trivially possible for any term expression and there is no maximum number of HLU Layers.

For a general encoding term expression, the feed-forward interconnection using the minimum number of HLU Layers may be found by selecting the best among all interconnections that produce the said term expression. Therefore, the minimum number of HLU Layers required determines a complete ordering of complexity of all possible HDC encoding expressions, and any HDC processor containing a limited number of HLU Layers is *qualitatively limited* in the class of HDC expressions it can encode. For example, a HDC processor with 8 HLU Layers cannot encode 11-grams. This is a unique property of the architecture developed here – it is unlike CPUs and GPUs where a smaller register files or on-chip cache increases dependence on off-chip memory but does not constrain the collection of possible programs that is computable.

4.1.3 The Valid Chain: a flow-based pipeline control

Section 3.2.1 abstracts the archetypal HDC processor that consumes at most 1 symbol each clock cycle, irrespective of the number of channels required for the workload. This is a good abstraction to design a widely programmable HDC processor. However, following the development of the Encoder's architecture, a few details can be added to this abstraction. Since the HLU Layer Network architecture establishes the Encoder as a pipelined structure, a signalling scheme at the processor's input interface must be developed to support the three fundamental functions of a pipeline control logic:

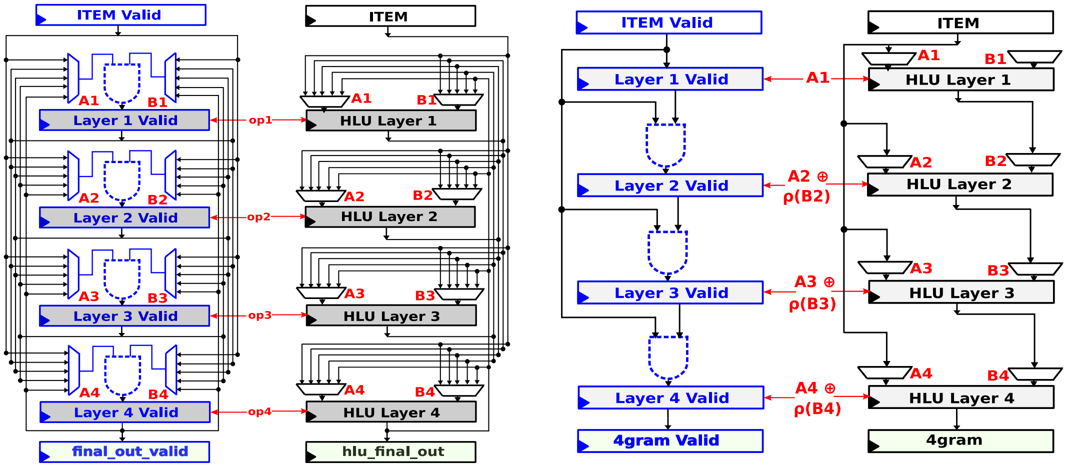
1. **Forward progress:** Fill pipeline with incoming symbols correctly from an initial state after a system reset or power-up. During live computation, fill the pipeline stages with newer results in a correct order, replacing the stale results of previous steps. An important part of forward progress is to prevent **deadlock** i.e. to ensure that the pipeline never stops computation while a valid input is supplied to it each cycle and no hardware faults or exceptions have occurred.
2. **Retention:** Hold intermediate results in place at pipeline stages when no symbols are available but the input stream or computation hasn't ended.
3. **Flush on completion:** When the end of computation is signalled, the pipeline stages still retaining intermediate results are emptied in an orderly manner and consumed to finish encoding, after which Encoder moves into a final "end" state.

Between the end of a computation and beginning of the next, it may be assumed that the processor's control logic resets the Encoder with an effect identical to that after a fresh power-up. Any results that need to be shared between different computations are therefore assumed to be stored in an Item or Associative Memory.

Forward progress, retention and pipeline-flush on completion must be implemented by *any* pipelined system. However, most pipelined structures usually have fixed stage-dependency patterns which greatly simplifies its control logic. For example, the classical single-issue, 5-stage pipeline for a scalar, in-order CPU has a fixed order of forward progress for instructions, a fixed logic to determine stalls due to hazards, and a fixed pattern of handling exceptions and completion – all *irrespective of the actual instruction stream being executed*. Contrast this with the HLU Layer Network in a Generic encoder, where changing the dependency pattern among stages is the principal manner in which different encoding expressions are programmed. Therefore, its control logic must be able to discharge the 3 main functions for a large variety of stage-dependencies possible in any reasonably capable Encoder. A simple way to handle such high complexity is to adopt a flow-based control for the pipeline as well – where programming the HLU Layers' dependency must simultaneously program the pipeline control as well. Thus, pipeline control emerges as a result of a *flow of control state* through it, identical to the way data is programmed to flow through its stages. This is illustrated in figure 4-4.

The pipeline control system is developed as follows. First, the following four fundamental control states define meaning of the contents of all HLU Layers:

1. `INVALID (X)`: The state of all HLU Layers immediately after a reset. This is also the valid signal representing a pipeline stall when stages are supposed to retain their data until the next valid datum arrives.



(a) Valid Chain in fully-connected network

(b) 4-gram Valid Chain

Figure 4-4: Flow-based pipeline control in HLU Layer Network.
An example for (a) fully-connected network and (b) feed-forward interconnection for 4-gram with 4 HLU Layers is shown. The **data-path** and **flow-based pipeline control** portions have different colors. Control bits stored in each stage are called *valid signals*. `ITEM` and `hlu_final_out` are the input and output hyper-vectors for the HLU Network respectively; they too have valid signals to denote the nature of the present input and the currently encoded term hyper-vector. The dotted AND gate in the valid signal logic indicates that a stage's hyper-vector should be set as valid for computation if and only if its operands were (both) valid.

2. `VALID (V)`: asserts that the values stored in the corresponding HLU Layer is a valid intermediate hyper-vector, contributing in the encoding process.
3. `DELIMIT (D)`: asserts that the HLU Layer is being emptied (i.e. the its vector stored is converted to all zeros, the default state after a reset). This does not signify that the current encoding session has ended. The HLU Layer may still be used to produce meaningful hyper-vectors for the encoding.
   When the `DELIMIT` state reaches the last stage of the HLU Layer Network, the shift-reg `ENABLEREG` of the accumulator is reset. No changes are made to the accumulator's values.
4. `END (E)`: asserts that the HLU Layer has completed computation and no longer contains or will produce a hyper-vector useful to the current encoding process.

To simplify the pipeline control logic, it is useful to impose a priority order among the 4 fundamental control states, which can be inferred for the meaning they represent. In particular, completing the computation through the pipeline stages once the end signal is received is more important than any ongoing pipeline flush procedure, which in turn is more important that the normal progress of data through the pipeline. A result produced by an HLU Layer is valid if and only if (both) its operand(s) is(are) simultaneously valid – the pipeline control logic is represented by a dotted *AND* gate in fig. 4-4) to emphasize this. Therefore, the priority order is

$$\texttt{INVALID (X)} > \texttt{END (E)} > \texttt{DELIMIT (D)} > \texttt{VALID (V)}$$

One may find the highest priority accorded to INVALID (X), the default state after reset, rather curious. However it is the natural choice as the pipeline should remain in reset state if no valid input vector has been asserted. Similarly, a DELIMIT (D) or END (E) assertion would not also not travel through the pipeline as emptying or ending a pipeline after reset is meaningless. As an immediate consequence of the priority order above, forward progress of valid data is *impossible* if feedback is present in interconnections. For example, for the feedback shown in figure 4-3(b), the pipeline control logic always produces the valid signal $max(\texttt{X}, \cdot) = \texttt{X}$ for the HLU Layer.

Assuming the convention that single-operand operations of delay and permute have the unused operand with the lowest priority state VALID (V), the pipeline control logic can be formulated as

$$\texttt{valid}[out] = max(\texttt{valid}[A], \texttt{valid}[B])$$

This is sufficient to guarantee forward progress and pipeline flush in feed-forward interconnections. It can be easily seen that the feed-forward property allows forward progress of data over the reset state INVALID (X), of newer data over old data (since VALID (V) remains asserted), and of pipeline flushes and end of encoding signals, as dictated by their priority ordering. An example for a feed-forward implementation of the 4-gram is shown is figure 4-6.

However, data retention and resumption of encoding is not supported as shown for the feed-forward 4-gram interconnection in figure 4-7. While the lack of forward progress with feedback is manageable as all known HDC algorithms have feed-forward encodings as well, the data retention issue must be necessarily rectified.

A solution can be gleamed from the observation that INVALID (X) > VALID (V) is helpful for supporting forward-progress during normal operation, but the opposite relationship is required for the correct order of resuming operations after retention. Therefore, a new state similar in meaning to INVALID (X) is required, which shall be designated ACTIVE (A). The priority order for the 5 **pipeline control states** is

$$\texttt{INVALID (X)} > \texttt{END (E)} > \texttt{DELIMIT (D)} > \texttt{VALID (V)} > \texttt{ACTIVE (A)}$$

The key here is to specify that the input vector to the pipeline i.e. the ITEM in fig. 4-4 can never have the ACTIVE (A) control state. The convention for single-operand operations of delay and permute is modified for the unused operand to have the lowest priority state ACTIVE (A). The pipeline control logic needs modification: an improved logic is specified as a finite automaton in figure 4-5.

This solves the data retention issue for feed-forward networks, as illustrated for 4-grams in figure 4-8. One can readily verify that forward progress and pipeline flush are supported for feed-forward interconnections as well (for example, 4-grams in fig. 4-6). The 5-stage Valid Chain logic is adopted in the remainder of this dissertation. Remember that under this scheme, although the Valid Chain logic of the HLU Layers

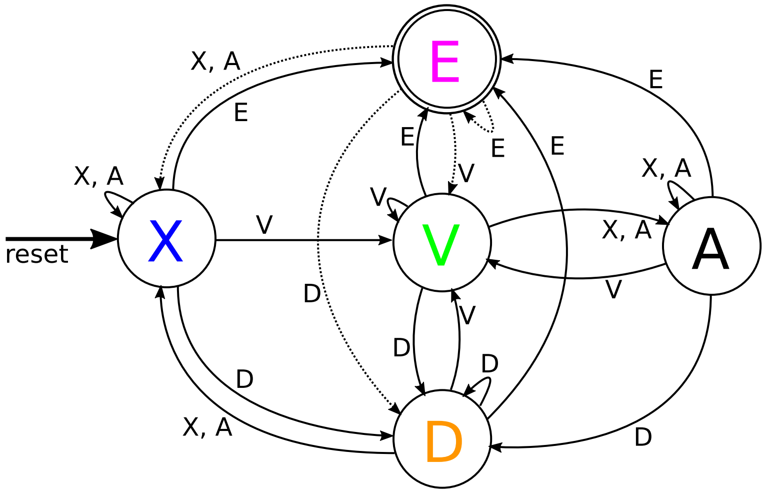
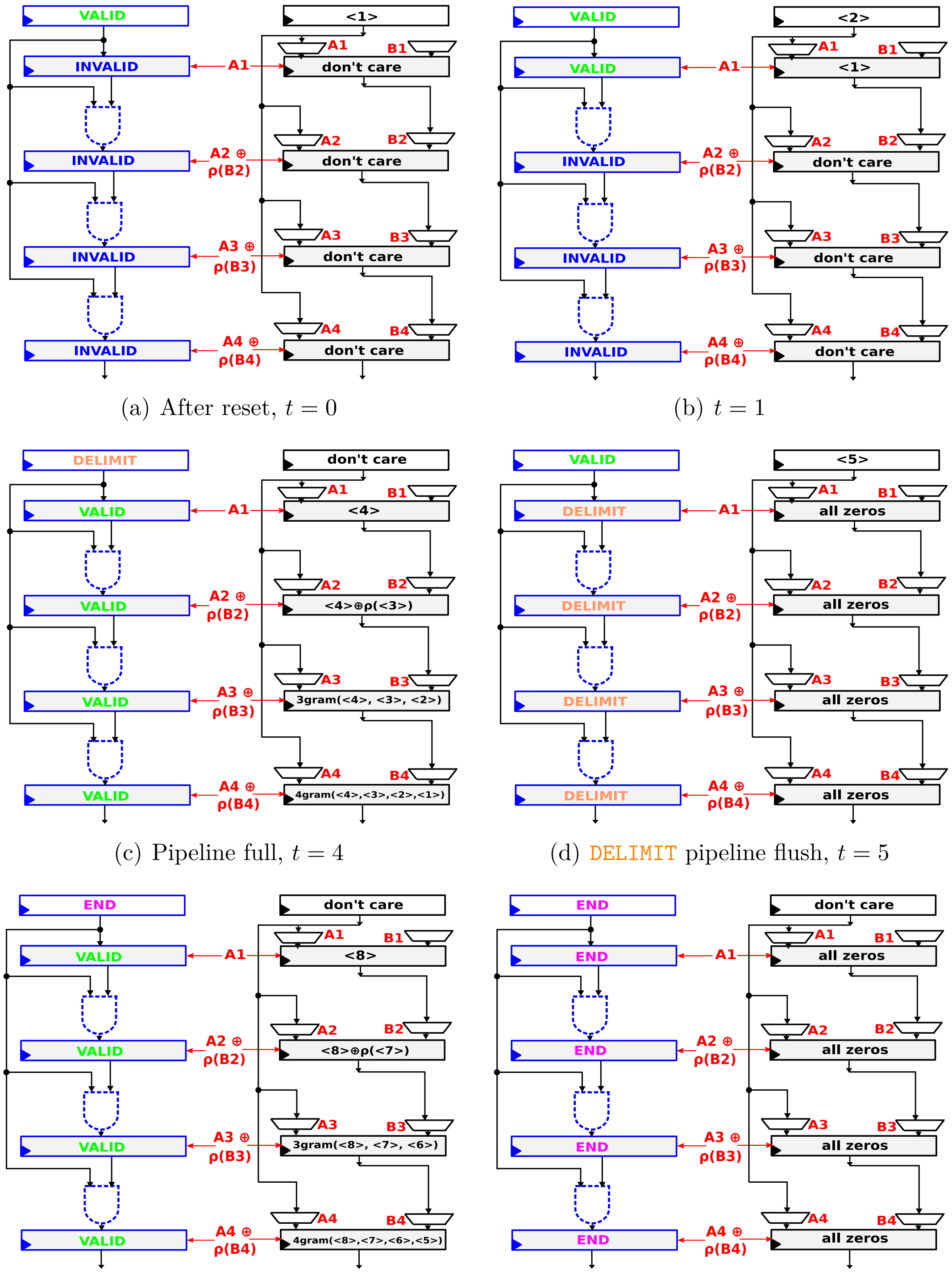


Figure 4-5: State transition diagram for `valid`$[out]$ using the 5-signal pipeline control. The 5 states are `INVALID (X)`, `END (E)`, `DELIMIT (D)` `VALID (V)` and `ACTIVE (A)`. On reset HLU Layer goes to state `X`. Doubly-circled `E` is the end state. Transition edges are marked by the causal $max($`valid`$[A],$`valid`$[B])$ from operands A, B.

use 5 valid signals, the *input to the HLU Layer Network pipeline still uses the earlier 4 valid signals*: `INVALID (X)`, `END (E)`, `DELIMIT (D)` and `VALID (V)`.



(a) After reset, $t = 0$

(b) $t = 1$

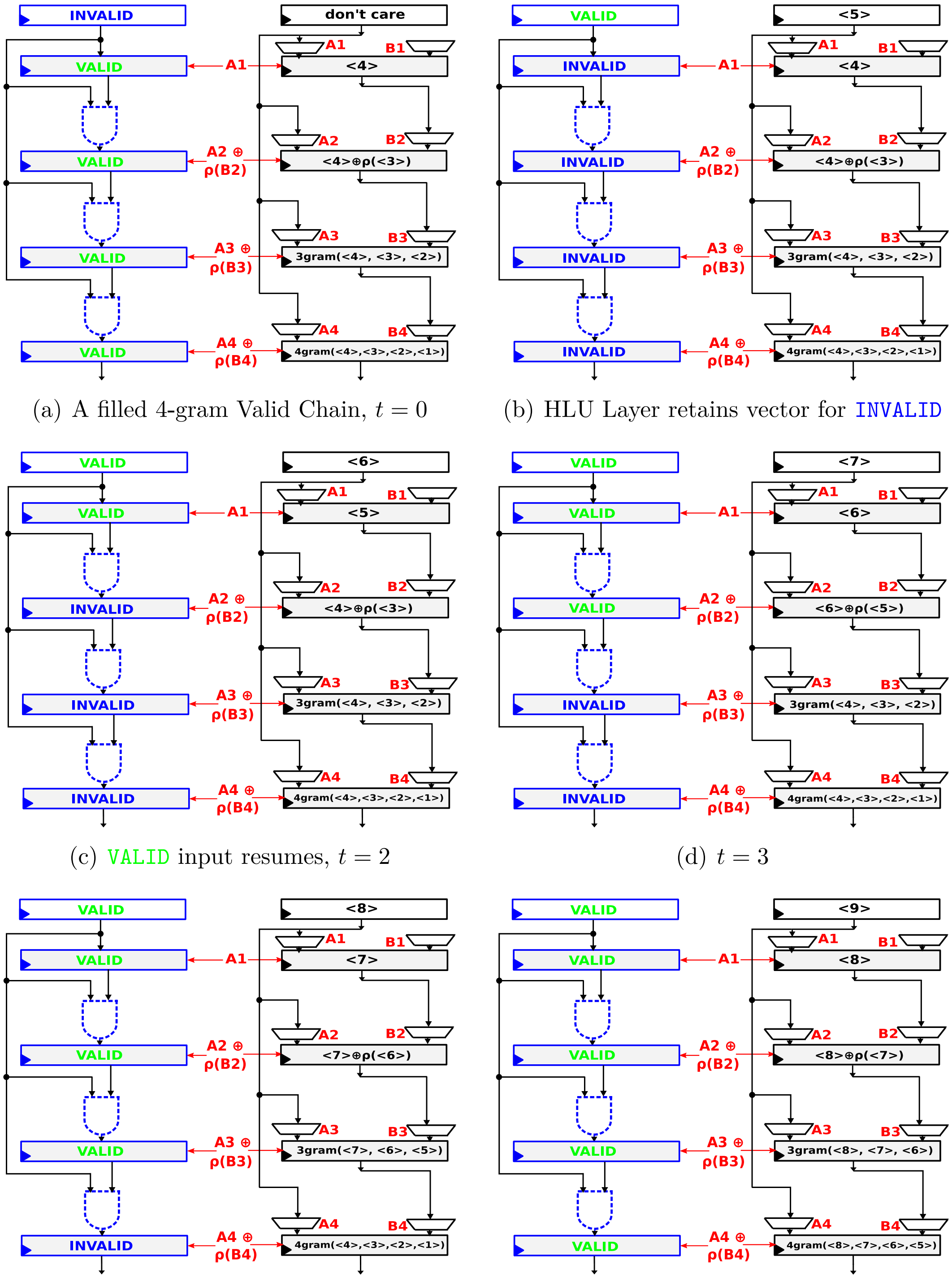
(c) Pipeline full, $t = 4$

(d) DELIMIT pipeline flush, $t = 5$

(e) Pipeline re-filled, $t = 9$

(f) END of encoding, $t = 10$

Figure 4-6: The basic 4-signal pipeline control is sufficient for forward progress and pipeline flush in feed-forward networks.
Hyper-vectors $< 1 >, < 2 >, \ldots, < 8 >$ are inputted in sequence; steps (a) - (f) show the evolution for a feed-forward 4-gram interconnection. Encoding ends when the last stage has state END. Short hands $\text{3gram}(a, b, c) \triangleq a \oplus \rho(b) \oplus \rho^2(c)$ and $\text{4gram}(a, b, c, d) \triangleq a \oplus \rho(b) \oplus \rho^2(c) \oplus \rho^3(d)$ are used for brevity.



(a) A filled 4-gram Valid Chain, $t = 0$

(b) HLU Layer retains vector for `INVALID`

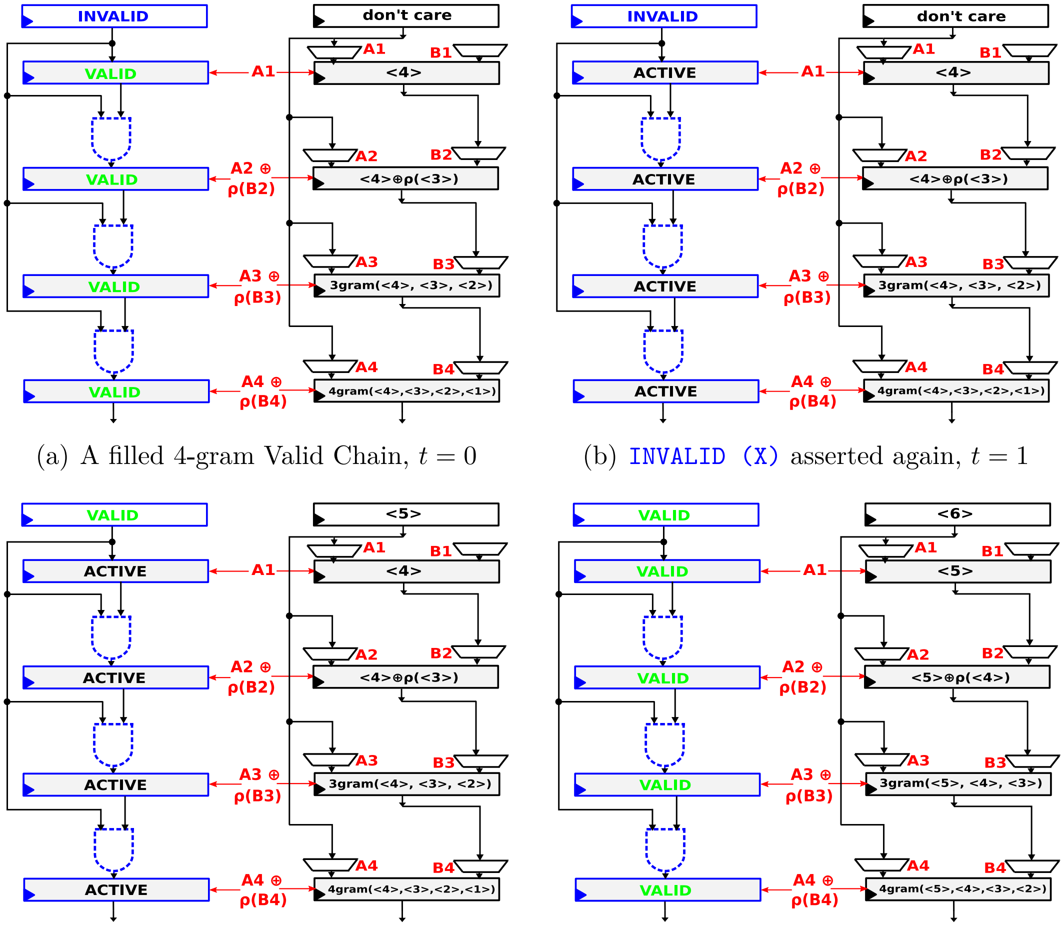
(c) `VALID` input resumes, $t = 2$

(d) $t = 3$

(e) $t = 4$

(f) Pipeline re-filled, $t = 5$

Figure 4-7: The basic 4-signal pipeline control does not support data retention. Short hands $\text{3gram}(a, b, c) \triangleq a \oplus \rho(b) \oplus \rho^2(c)$ and $\text{4gram}(a, b, c, d) \triangleq a \oplus \rho(b) \oplus \rho^2(c) \oplus \rho^3(d)$ are used for brevity. Hyper-vectors $< 1 >, < 2 >, \ldots, < 9 >$ are inputted in sequence; the term expressions $\text{4gram}(< 5 >, < 4 >, < 3 >, < 2 >)$, $\text{4gram}(< 6 >, < 5 >, < 4 >, < 3 >)$ and $\text{4gram}(< 7 >, < 6 >, < 5 >, < 4 >)$ are skipped due to the intervening `INVALID` input.



(a) A filled 4-gram Valid Chain, $t = 0$

(b) `INVALID (X)` asserted again, $t = 1$

(c) Data retention for `ACTIVE (A)`, $t = 2$

(d) `VALID (V)` fills pipeline on resume, $t = 3$

Figure 4-8: 5-signal pipeline control supports data retention for feed-forward networks. For the feed-forward 4-gram network, (a) - (d) shows the pipeline control when two `INVALID (X)` inputs are asserted before the next `VALID (V)` input. Unlike in figure 4-8, no 4-grams are missed here as the *entire pipeline* is live immediately after the subsequent `VALID (V)` input is asserted at $t = 2$.

4.1.4 Considerations of sparsity and security

Classically, all items used are *dense* i.e. their bits are equally likely to be a 0 or 1. However, it is known that *sparse* items – where the probability of a bit being 1 is a chosen (small) $0 < p \leq 1/2$, leads to far greater energy efficiency with negligible degradation in performance [117]. Therefore, it is important to evaluate the proposed generic architecture on its ability to gain benefits from **sparsity** of hyper-vectors. The generic Encoder architecture does not prevent the use of sparse hyper-vectors.

Security using hyper-vectors principally comes from the fact that it is very hard to factorize numbers of very high magnitude (i.e. its binary description has thousands of bits). This has been harnessed to develop a framework to support secure learning over multiple client-server links in a distributed manner [118]. For decisions pertaining the Encoder architecture, security from **side-channel attacks** is more relevant, where the aim is to limit information that can be gained by monitoring the *physical characteristics* of the system on which the job is run (and not the actual calculations in the job). Since each HLU Layer contains registers, they are easier to observe and monitor using heat maps, and may be connected to scan chains vulnerable to scan-based side-channels [119]. Consequently, it desirable to prevent a leak of any meaningful and interpretable hyper-vectors from any *intermediate* HLU Layer.

This short analysis demonstrates that sparsity and side-channel security are often interrelated and can be traded-off by choosing a different encoding interconnection. Consider again the feed-forward and feedback interconnections for 3-gram in figure 4-3(a), (c). Let `ITEM` X_t for all time-steps $t > 0$ have sparsity p i.e. all its bits have probability p of being 1. We are interested in calculating the sparsity (probability of a bit being 1) of all HLU Layers' outputs in figure 4-3(a), (c). For the feed-forward interconnection in (a), the HLU Layers progressively construct 2-gram $\triangleq X_{t-1} \oplus \rho(X_{t-2})$ followed by 3-gram $\triangleq X_{t-1} \oplus \rho(X_{t-2}) \oplus \rho^2(X_{t-3})$. The HLU Layer in (b) and HLU Layer 1 in (c) produces $Z_t \triangleq X_{t-1} \oplus \rho(X_{t-2}) \oplus \rho^2(X_{t-3}) \oplus \rho^3(X_{t-4}) \oplus ...$ as defined in section 4.1.2, and intermediate HLU Layers $2, 3, 4$ in (c) simply delay Z_t.

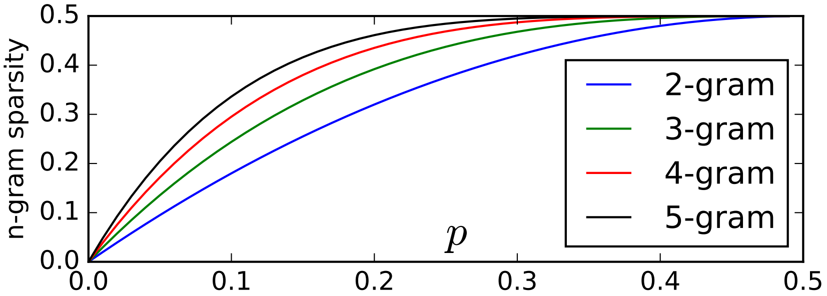


Figure 4-9: Sparsity of n-gram with item sparsity.
Sparsity is the probability of a bit being 1. Observe that n-gram sparsity increases towards 1/2 with increasing n for all item sparsity $0 < p \leq 1/2$.

Hence the sparsity of n-grams is required here, for $n = 2, 3$ and $n = \infty$ (for Z_t). This calculation is simplified by using bipolar bits with value -1 instead of 0. From the i.i.d. nature of bits in an item, the bit-wise XOR in $X_{t-1} \oplus \rho(X_{t-2})$

does XOR of two independent bits. For a bipolar bit x with sparsity p we have that $\mathbb{E}[x] = 2p - 1$. The XOR function for bipolar vectors is $x \oplus y = -xy$. Thus, the sparsity of $x \oplus y$ is $\Pr[x \oplus y = 1] = \frac{1}{2}(1 - \mathbb{E}[xy])$; when x and y are independent $\Pr[x \oplus y = 1] = \frac{1}{2}(1 - \mathbb{E}[x]\,\mathbb{E}[y])$. More generally, for independent bipolar bits $x_1, x_2, \ldots, x_n$ with sparsity $0 < p \leq 1/2$, the sparsity of n-grams equal sparsity of $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ which is $\Pr[\oplus_{i=1}^{n} x_i = 1] = \frac{1}{2}(1 - (-\mathbb{E}[x_1])^n) = \frac{1}{2}(1 - (1 - 2p)^n)$. Since $0 < p \leq 1/2 \implies 0 \leq 1 - 2p < 1$, conclude that sparsity of n-gram given by $1/2 - (1 - 2p)^n/2$ increases to $1/2$ as $n \uparrow \infty$. Sparsity of n-grams for $n = 2, 3, 4, 5$ is shown in fig. 4-9 for item sparsity $0 < p \leq 1/2$.

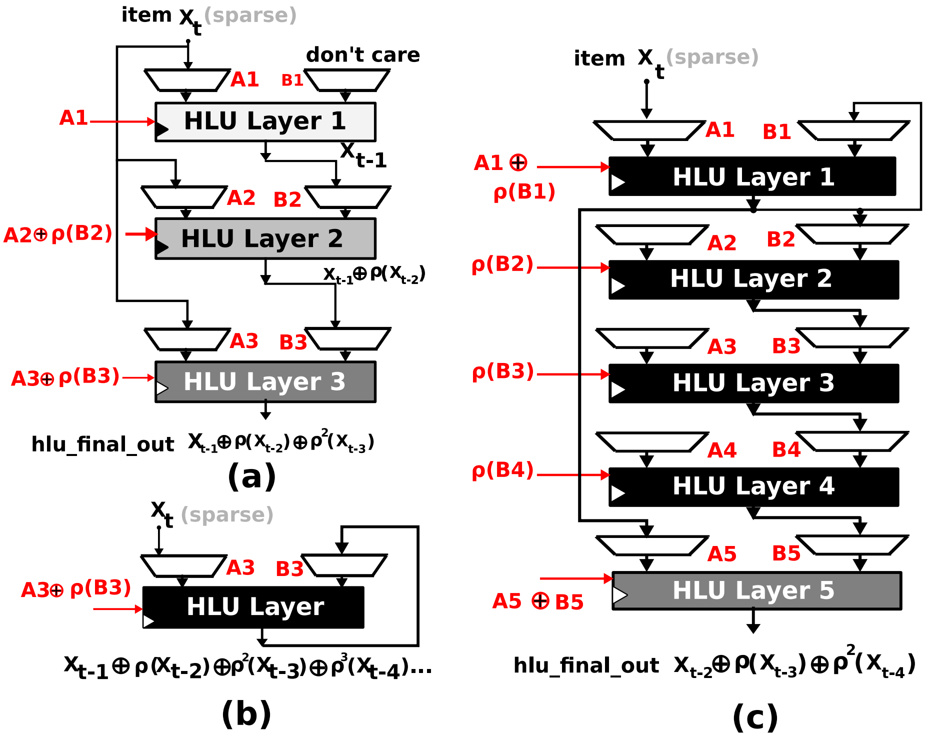


Figure 4-10: Trade-off between sparsity and side-channel security for 3-gram. Fig. 4-3 has been re-annotated with the varying degrees of sparsity of (the outputs by) HLU Layers when inputs X_t are sparse hyper-vectors. A darker color HLU Layer represents that its output hyper-vector is denser. The black HLU Layers of (b) and (c) represent that these HLU Layers always output dense hyper-vectors. Thus, HLU Layers in (a) are sparser but risk leaking interpretable 2-grams and 3-grams, whereas (b) and all except last HLU Layer in (c) produce dense vectors which are garbled.

One can now see how sparsity and security are inter-related for 3-grams. Figure 4-10 is a sparsity-annotated version of fig. 4-3. For the feed-forward network, (a) has a sparser but meaningful intermediate vector which can be used to leak the 2-gram of input items. However, feedback in (b) allows (c) to have dense but secure intermediate hyper-vectors, independent of item sparsity p. This is because Z_t binds all items consumed by the processor and is completely garbled for any finite-n n-gram encoding.

4.2 Extensions of the Generic architecture

To summarize all architectural features developed to far, it is instructive to see how the various components work together in a complete HDC system to handle a known HDC algorithm. For simplicity of the architecture as sketched in fig. 3-5, a Read-Only Memory (ROM) consisting of a fixed set of pseudo-random vectors generated only once is considered as the Item Memory, and the Associative Memory as shown in the figure is composed of flip-flops and digital standard cells only. Although most HDC encodings are single-stage, EMG hand-gesture recognition [88, 39] is a prominent application requiring two stages. Hence, consider a two-stage encoder capable of training and testing EMG hand-gestures. Recall that the encoding expression is a superposition of 4-grams from 64 features [39] – the first stage computes a superposition of channel-values pair bindings from the measurements of 64 channels; on its completion the second stage computing 4-grams of results from the first stage advances by a pipeline stage. The generic HDC processor consuming a symbol every cycle (as defined in section 3.2.1) can read in an EMG sample as an input stream by consuming either a channel hyper-vector or that of its value with an accompanying valid signal.

Perhaps the first thing one comes to ponder about the two-stage generic Encoder is: how does one control the encoding across two stages, each of which require valid signals *locally* for conveying to their HLU Layers about the `END (E)`, `DELIMIT (D)`, `VALID (V)` and `INVALID (X)` control states? In particular, how does one implement the two stage-pipeline controls and integrate them such that the Valid Chain for each stage is the same as described in section 4.1.3, but when configured to work together the second stage progresses by one (due to a local `VALID (V)`) whenever the first stage completes encoding (a role for local `END (E)`)? Clearly, even though the processor as a whole receives a *single input valid signal* from a common input interface, the same valid signal cannot be used for both stages in a clock cycle. A translation logic intervening between the two stages is necessary. A possible solution is to have a new designation of the terminating state, called `EE`, with a higher priority than the default `E` (i.e. `X` > `EE` > `E` > `D` > `V` > `A`). Figure 4-11 shows a Generic processor with a two-stage Encoder – the 5 (input) valid signals with an extra `END` state and all Valid Chain logic that uses it are marked with an asterisk(*). `E` and `EE` are treated identically by the first stage `E1`. Two Logic blocks `Logic X` and `Logic Y` translate the 5-signal (input) valids from the input interface to the 4-signal (input) valid as developed in section 4.1.3 before consumption by the second stage `E2`. If the first stage is unused, only `EE` is translated to `E` and all other valid signals are unchanged. If `E1` is used, valid signals are translated by `Logic Y` as shown in fig. 4-11.

For two-stage encoding data like EMG, it is important to correctly employ the `DELIMIT (D)` signal and the higher priority `END (EE)` signal to effect advancement of `E2` by a cycle and to end encoding. This is shown in figure 4-12.

Finally, to demonstrate the need for distinction between `E` and `EE`, consider the problem of recognizing the emotional state from physiological measurements. **Emotion Recognition** has also seen a successful application of two-stage HDC encoding [120], where 4-grams of 32-feature galvanic skin response measurements (GSR), 77-feature electrocardiogram or heart-rate measurements (ECG) and 105-feature electroencephalo-

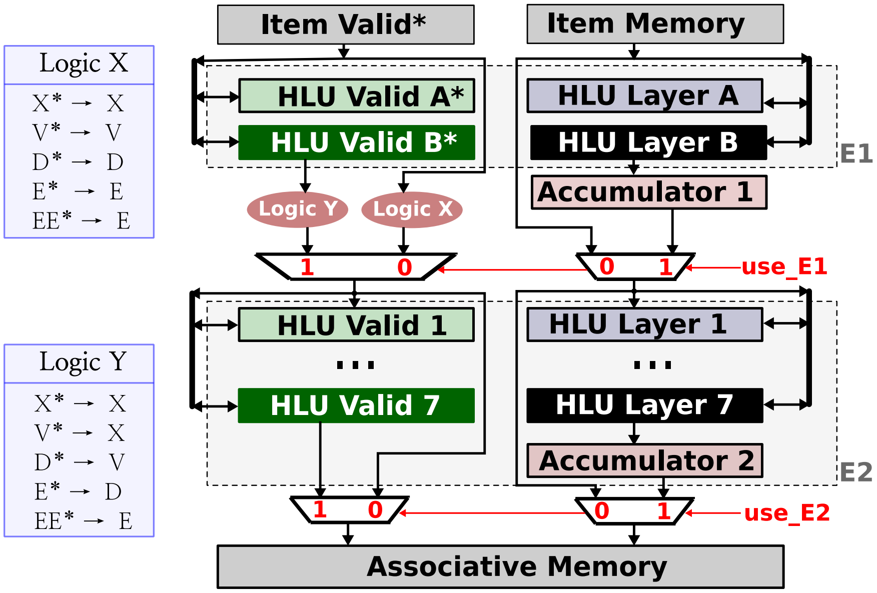


Figure 4-11: Two-stage encoder in a Generic HDC processor. The Encoder is split into two stages: Accumulator 1, HLU Layer A and HLU Layer B forms the first stage E1, and Accumulator 2, HLU Layer $1-7$ forms the second stage E2. HLU Layer B and HLU Layer 7 are the output layers of E1 and E2 respectively. At each clock cycle, the input to Item Memory is provided with a valid signal from a 5-signal collection $\{\mathtt{X}^*, \mathtt{V}^*, \mathtt{D}^*, \mathtt{E}^*, \mathtt{EE}^*\}$, where $\mathtt{EE}^*$ is an END state of higher priority than $\mathtt{E}^*$ – to allow signalling for two-stage encoding. Item Valid, HLU Valid A and HLU Valid B, marked by asterisk(*), have a valid logic that accept values from the 5-signal collection. HLU Valid $1-7$ admits valid signals from the 4-signal valid set $\{\mathtt{X}, \mathtt{V}, \mathtt{D}, \mathtt{E}\}$ only. Logic X and Logic Y converts valid signals from $\{\mathtt{X}^*, \mathtt{V}^*, \mathtt{D}^*, \mathtt{E}^*, \mathtt{EE}^*\}$ to $\{\mathtt{X}, \mathtt{V}, \mathtt{D}, \mathtt{E}\}$ depending on whether first stage E1 is used.

gram or brain-wave measurements (EEG) are superimposed together to form the encoded hyper-vector. Unlike EMG hand-gesture recognition, Emotion recognition needs the second-stage 4-gram encoding pipeline to flush once when switching from GSR to ECG data and again from ECG to EEG data. To support flushing (but *not completion*) of the second stage of encoding by the propagation of a local DELIMIT signal, the E valid signal is used at the input interface. The complete input scheme of the physiological data for Emotion recognition is illustrated in figure 4-13.

This completes the description of the Generic architecture. The following subsections consider extensions of the generic architecture developed so far. Along with increasing the number of Encoders, Item and Associative memories, various implementation choices and advanced control structures are discussed.

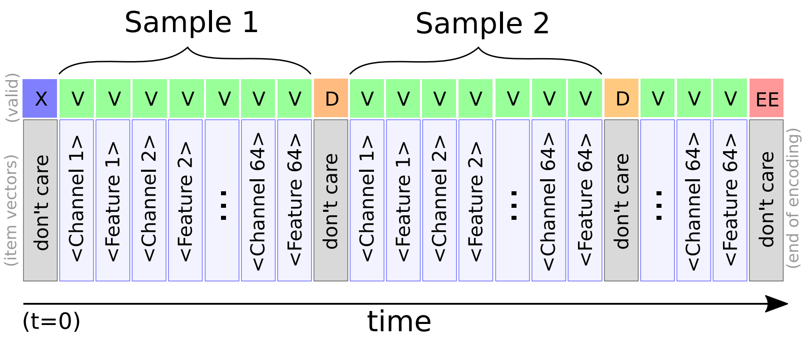


Figure 4-12: Valid signals and input scheme for two-stage encoding of EMG hand-gesture data in a Generic HDC processor.
The DELIMIT (D) valid signal is used between two 64-feature frames and the higher priority END (EE) is used to signal end of encoding.

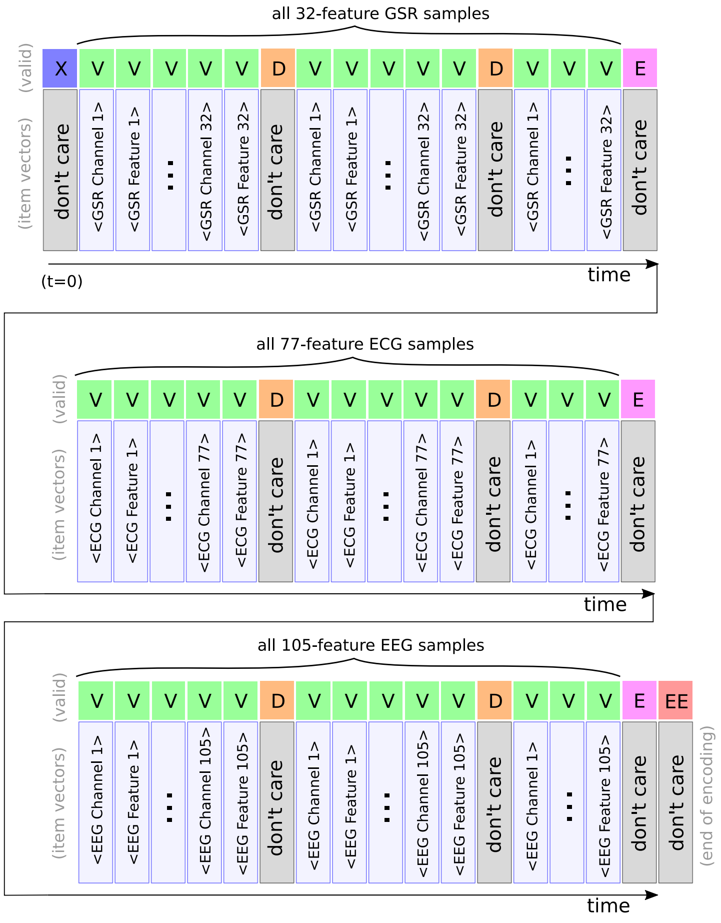


Figure 4-13: Valid signals and input scheme for two-stage encoding of physiological data for Emotion recognition in a Generic HDC processor.
The DELIMIT (D) valid signal is used between two feature frames, the lower priority END (E) is used to switch from GSR to ECG and then to EEG samples, and the higher priority END (EE) is used to signal end of encoding.

4.2.1 Item Memory and its extensions

While figure 3-5 shows the Item Memory simply as a structure to store random hyper-vectors at fixed address locations, several extensions are useful to have. For vectors which do not need to be almost-surely orthogonal due to high dimensions – such as for the hyper-vectors representing scalars from an ordered field – it is more efficient to calculate them on-the-fly than use memory resources.

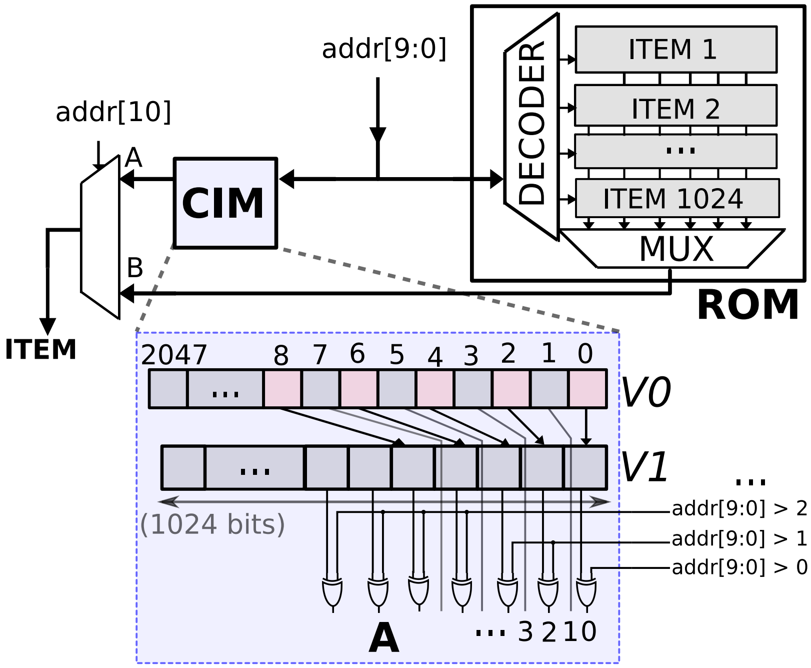


Figure 4-14: Item Memory with continuous-item generation logic for scalar values. An Item Memory with 2048 addresses is shown – where the lower half of addresses fetch a vector stored in a Read-Only Memory (ROM) and the upper half containing 1024 addresses produce hyper-vectors for scalars. The scalar value is quantized and mapped to the set of numbers $0 - 1023$. The hyper-vector `V0` is designated for scalar 0, loaded from the Read-Only Memory (ROM) during configuration, and the hyper-vector for scalar 1023 differs from `V0` at all even bits (i.e. exactly orthogonal to `V0`). The Continuous Item Memory (CIM) uses the value of the lower 9 bits address to determine number of even bits of `V0` to be flipped.

Continuous Item Generation: Assigning orthogonal vectors to integers or values from an ordered set may not be appropriate. Ideally, two close numbers should have a correspondingly strong correlation among their hyper-vector bits. A possible solution (by [87]) is to assign points on a line connecting two exactly orthogonal vectors. Then, any collection of 3 hyper-vectors A, B, C (representing integers $a \leq b \leq c$) will satisfy the triangle law $d_H(A, B) + d_H(B, C) = d_H(A, C)$. Equivalently, one can begin with a random vector `V0` and a direction vector `Y` with $(d/2)$-bits being 1. The smallest integer, usually 0, is mapped to `V0` and the largest integer, say M, is mapped to `V0` $\oplus$ `Y` (Fig. 4-14). For all other integers n, flip $\frac{d}{2M}$ additional bits per integer value along direction `Y` from the hyper-vector assigned to 0 – the only restriction is that M divides $d/2$. Clearly, the generation of continuous items is very cheap in logical effort and time: only an array of d XOR gates and an extra clock cycle in latency is added. An example is shown if figure 4-14 for $d = 2048, M = 1023$ and the direction vector Y having 1 in even bit positions.

Representing scalars and elements from other vector fields is a rapidly growing field of enquiry in the HDC community – section 3.2 in [64] provides a recent review of all published works in this space. A detailed theoretical treatment for computing on a reproducing kernel Hilbert space of real-valued functions using hyper-vectors is presented in [121].

Pseudo-random and random generation of items. The principal mandate of an Item Memory is to produce random hyper-vectors, the common starting point of all HDC algorithms, and to map it to the symbol set of the problem at hand (as discussed in sec. 3.2.1), storing this map for the duration of the problem. Therefore, its role is that of generation *and* storage. When implementing Item Memory with a Read-Only Memory (ROM) [32], the generation of vectors is performed only once by external means – only the choice of address-symbol mapping by `HD MAPPER` like in fig. 3-5 remains. This rudimentary strategy limits the number of items and fundamentally lacks entropy – a cornerstone of HDC's mechanism. Therefore, a natural choice is to look at on-chip generation of random hyper-vectors.

Pseudo-Random Number Generators (PRNGs) rely on deterministic calculations to produce bit-strings that *appear* random i.e. they are indistinguishable from independent and uniform bits with respect to the results of a collection of statistical tests. Multiple batteries of statistical tests are used to evaluate PRNGs, where the easy `ENT` suite [122], the `NIST` suite [123] and the difficult `DIEHARDER` suite [124] are common. Among the few PRNG algorithms known to pass these suites successfully, the "Mersenne Twister" algorithm (both the 32-bit `MT19937-32` and 64-bit `MT19937-64` variants) [125] is the most well known: it is part of the standard library of widely used programming languages including `python`, `C`, `C++11` and `MATLAB`. While a digital CMOS implementation of the Mersenne Twister requires only barrel shifters, AND and XOR gates, the PRNG suffers from a relatively large state buffer of 20480 bits and a low throughput of the order of 10^{-2} hyper-vector each cycle (it can produce at most 64 bits/cycle). The `TinyMT` variant is an attractive alternative as it uses only 127 bits of state but has far smaller period [126] ($2^{127} - 1$ instead of $2^{19937} - 1$ for Mersenne Twister, which is still reasonably large). Similarly, the `SFMT` variant producing 128 bits/cycle [127] and data-parallel variants particularly well-suited for architectures exploiting SIMD parallelism [128] are great candidates for exploring on-chip generation of pseudo-random vectors with reasonable throughput. A design exploration of Item Memory containing such PRNG is a great direction for fruitful research.

Cellular automata have been proposed as hardware-efficient PRNGs [129, 130] to be distributed in common standard libraries [131]. However, research suggests they are usually not very cryptographically secure as the effective key length is much smaller [132] and often fail relatively easy statistical tests (such as the chi-squared test [133]). While a few recent works have proposed cellular automata (especially rule 90) for hyper-vector generation in HDC [134, 135], the impact of likely lower entropy on HDC performance hasn't been studied yet. Cellular automata-based hyper-vector generation exhibits variable latency to produce hyper-vectors [135] (the latency depends on the address) which complicates the overall processor architecture. An in-depth analysis of cellular automata as a candidate for hyper-vector generation is a fertile area of study.

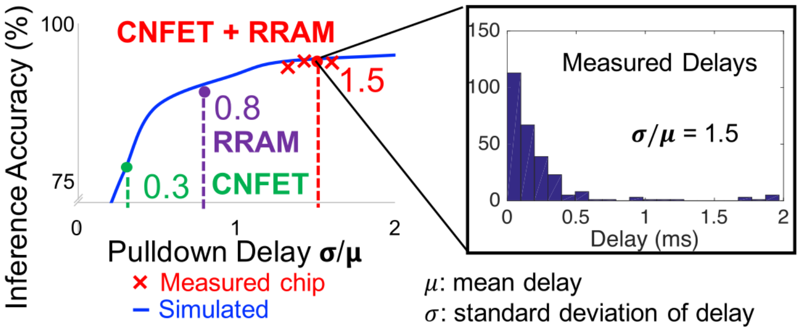
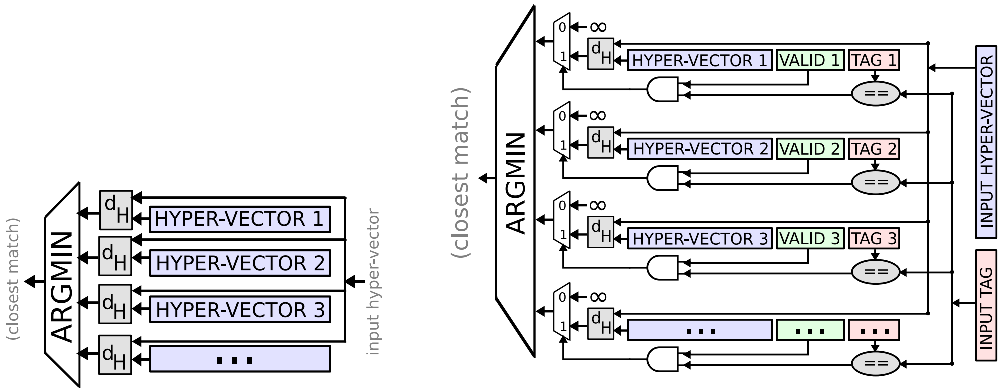


Figure 4-15: Figure 6 of [136]: Variation in cell delay is used to produce 27 items. Higher standard deviation σ per average delay μ indicates greater randomness. $\sigma/\mu = 1.5$ is sufficient as greater values produce diminished gains in accuracy.

True Random Number Generators (TRNGs) harness randomness in physical devices to produce random bits. TRNGs have been widely used on-chip as high-quality sources of entropy, where chaos [137] and metastability [138] are common mechanisms for generating randomness. TRNGs are the only choice when high-quality entropy is needed [139] but they are difficult to design and expensive in silicon area. Consequently, imperfect TRNGs may suffice for hyper-vector generation. For example, [136] uses the variability of a resistive RAM-based delay cell (see figure 4-15) to produce 32 random bits every cycle (hundreds of cycles are used to process a hyper-vector). Similarly, a relatively high entropy source may be obtained by a simple extractor algorithm [140] from real-time galvanic skin response (GSR) measurements (as used in Emotion recognition [120]). A design of an Item Memory with on-chip TRNG and its comparison with alternatives will be very useful to evaluate all the choices discussed here.

4.2.2 Associative Memory and its extensions

The main complexity of associative memory is the implementation of distance calculation logic. For binary vectors, hamming distance $d_H(\cdot,\cdot)$ is the natural choice. Hamming distance calculation requires an array of d XOR gates, the population count and closest match logic – a parallel implementation computing the maximum number of XORs across HDC dimension d is likely to make it the critical path in the processor [32]. The simplest Associative Memory is shown in figure 4-16(a). However, one may easily extend this implementation, such as a design using tags to store and compare multiple hyper-vectors for different tasks simultaneously. In this implementation



(a) The basic Associative Memory.

(b) Extended Associative Memory with tags to store multiple models.

Figure 4-16: Associative Memory and its extensions.

shown in figure 4-16(b), only those hyper-vectors which are valid and match the tag of the input are compared to report the closest match.

Digital logic for the distance calculation and comparison may not be the most efficient implementation. Since address of the closest match is the only output, the distance calculation can tolerate significant errors. Analog techniques show great promise of efficiency over digital CMOS implementations [141] and may prove to be the most successful strategy in mature HDC processors.

Associative Memory for vectors of non-binary elements such as integers, real or complex numbers necessitates a fundamental change in data-path from their binary counterparts. The most prominent properties of them are the far greater memory and logic requirements. The next chapter considers the problem of finding an efficient and feasible design of the Associative Memory storing integer hyper-vectors and using cosine similarity for comparisons. An emerging direction of research is to study HDC using hyper-vectors with elements from finite fields, which promotes hardware innovation and ease of adoption [142].

4.2.3 Multi-component extensions

The basic architecture containing one Item Memory, Encoder and Associative Memory each, as shown in figure 3-5, may be scaled up to contain multiple instances of them.

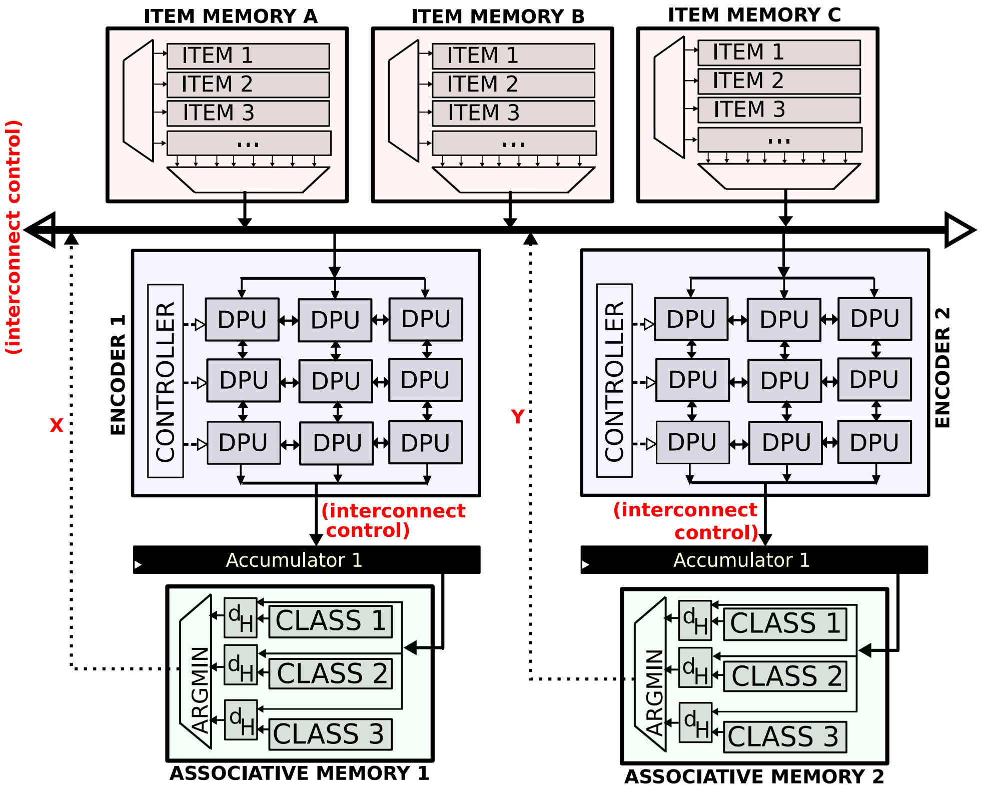
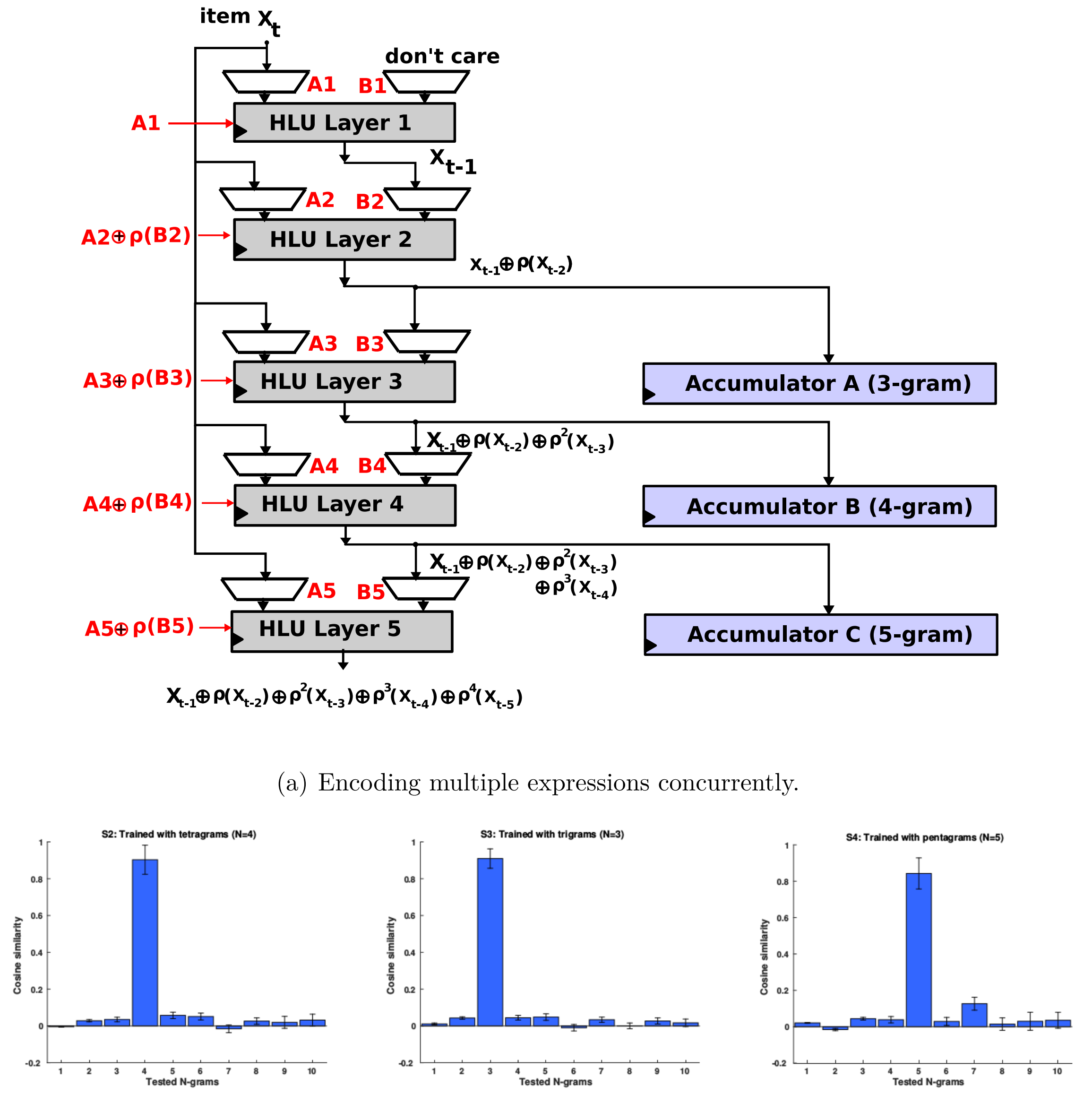


Figure 4-17: A scaled up HDC processor with multiple components. The uni-directional data-flow architecture may be extended to allow hyper-vectors to flow from Associative Memories back to Encoders (marked **X** and **Y**).

This is illustrated in figure 4-17. Multiple components interconnected in the same processor allow sharing of items and hyper-vectors, iterative HDC algorithms such as factorization [108, 143], and greater energy efficiency. Similarly, each HLU Layer in a HLU Layer Network of the Encoder can produce different encoding expressions simultaneously. An example for concurrently producing 3-grams, 4-grams and 5-grams from the common input stream is shown in figure 4-18.


item X_t
don't care
A1
B1
A1
HLU Layer 1
X_{t-1}
A2
B2
A2⊕ρ(B2)
HLU Layer 2
$X_{t-1}\oplus\rho(X_{t-2})$
A3
B3
A3⊕ρ(B3)
HLU Layer 3
Accumulator A (3-gram)
$X_{t-1}\oplus\rho(X_{t-2})\oplus\rho^2(X_{t-3})$
A4
B4
A4⊕ρ(B4)
HLU Layer 4
Accumulator B (4-gram)
$X_{t-1}\oplus\rho(X_{t-2})\oplus\rho^2(X_{t-3})$
$\oplus\rho^3(X_{t-4})$
A5
B5
A5⊕ρ(B5)
HLU Layer 5
Accumulator C (5-gram)
$X_{t-1}\oplus\rho(X_{t-2})\oplus\rho^2(X_{t-3})\oplus\rho^3(X_{t-4})\oplus\rho^4(X_{t-5})$
S2: Trained with tetragrams (N=4)
S3: Trained with trigrams (N=3)
S4: Trained with pentagrams (N=5)
Cosine similarity
Tested N-grams
1
0.8
0.6
0.4
0.2
0
-0.2
1 2 3 4 5 6 7 8 9 10

(a) Encoding multiple expressions concurrently.

(b) Figure 8 of [87]: EMG Hand-gesture recongition requires different n-grams for different subjects.

Figure 4-18: A common HLU Layer Network can be configured to simultaneously encode multiple expression from a common input stream.

Encoding a collection of HDC expressions concurrently can reduce total time and make the job more efficient without incurring performance loss. For example, EMG hand-gesture recognition often benefits from training multiple n-grams and choosing the best among them depending on the subject. This is especially useful when the number of channels is small – for example, when only 4 channels were available for EMG measurements in [87].

4.3 Hardware evaluation of Generic architecture

This section concludes the development of the generic HDC architecture with a summary of energy costs per inference for a suite of supervised classification tasks. These benchmark and results are described in detail in [32].

4.3.1 Benchmark of supervised classification tasks

Applications	**Abbrev.**	**Encoding**	**HDC**	**Known State-of-the-Art Algorithm**
Language Recognition	LANG	4-gram	90.6 %	97.1 %, n-gram-based Nearest Neighbors [19]
EMG Hand-Gesture Recognition*	EMG	2-stage	**95.8** %	89.7 %, Support Vector Machine [38]
Fetal State Classification (cardio.)*	CARDIO	21-features	**90.6** %	90.6 %, Support Vector Machine [144]
Page-block Classification	PAGE	10-features	**91.6** %	85.9 %, min-max Hyperplane Separation [145]
UCI Human activity Recognition*	UCIHAR	561-features	76.7 %	89.3 %, Support Vector Machines [146]
Spoken Letter Classification	ISOLET	617-features	75.9 %	97.1 %, boosted k-Nearest neighbors [147]
Human Face Detection*	FACE	608-features	66.0 %	96.1 %, HOG-based boosted Decision Trees [148]
MNIST Digit Classification	MNIST	784-features	75.4 %	99.7 %, Deep Convolution Neural Network [91]

Table 4.1: Benchmark for energy evaluation of the Generic HDC architecture. A collection of 8 supervised classification tasks with varying complexity were chosen to evaluate a generic HDC processor with $d = 2048$, an Item Memory with 2048 items, ROM and CIM as shown in figure 4-14, an Associative Memory with 32 class vectors, and a two-stage Encoder as with 2 and 7 HLU Layers respectively as shown in figure 4-11. (*)-members are well suited for human-centric computing in Internet of Things (IoT) as identified in [149].

To evaluate the Generic architecture, a set of applications must be chosen to faithfully represent the state of the art. The following 8 applications were a part of the benchmark chosen in Section III of the hardware evaluation study in [32].

Language Recognition (LANG) is described in section 2.3.2. EMG Hand-Gesture Recognition (EMG) classifies electromyography signals recorded from a subject's hand into a set of hand-gestures – described in section 3.1.1. Fetal State classification (CARDIO) uses measurements of heart-rate and uterine pressure during pregnancy to classify fetal condition before delivery [144]. Page-block classification (PAGE) finds all blocks of the page layout in a document that has been detected by a segmentation process [145]. UCI Human-activity Recognition (UCIHAR) classifies recordings of 30 subjects performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors [146]. Spoken Letter Classification (ISOLET) predicts the English letter spoken from voice recordings of subjects. Face Detection (FACE) determines whether a human face is present within a given picture frame [148]. MNIST Digit Recognition (MNIST) classifies the digit from images of drawn digits [91] – described in section 3.1.1.

Table 4.1 compares the accuracy of single-pass HDC with the best known ML models for each benchmark dataset from the literature. **Bold** indicates better or equal accuracy for HDC over the best-known ML algorithm. The list is non-exhaustive but contains representative datasets from human-centric IoT summarized in [15]. It is also *balanced* overall: MNIST, FACE and ISOLET represent the fact that known HDC algorithms alone applied on raw features are not competitve compared to ML for even simple speech and vision problems.

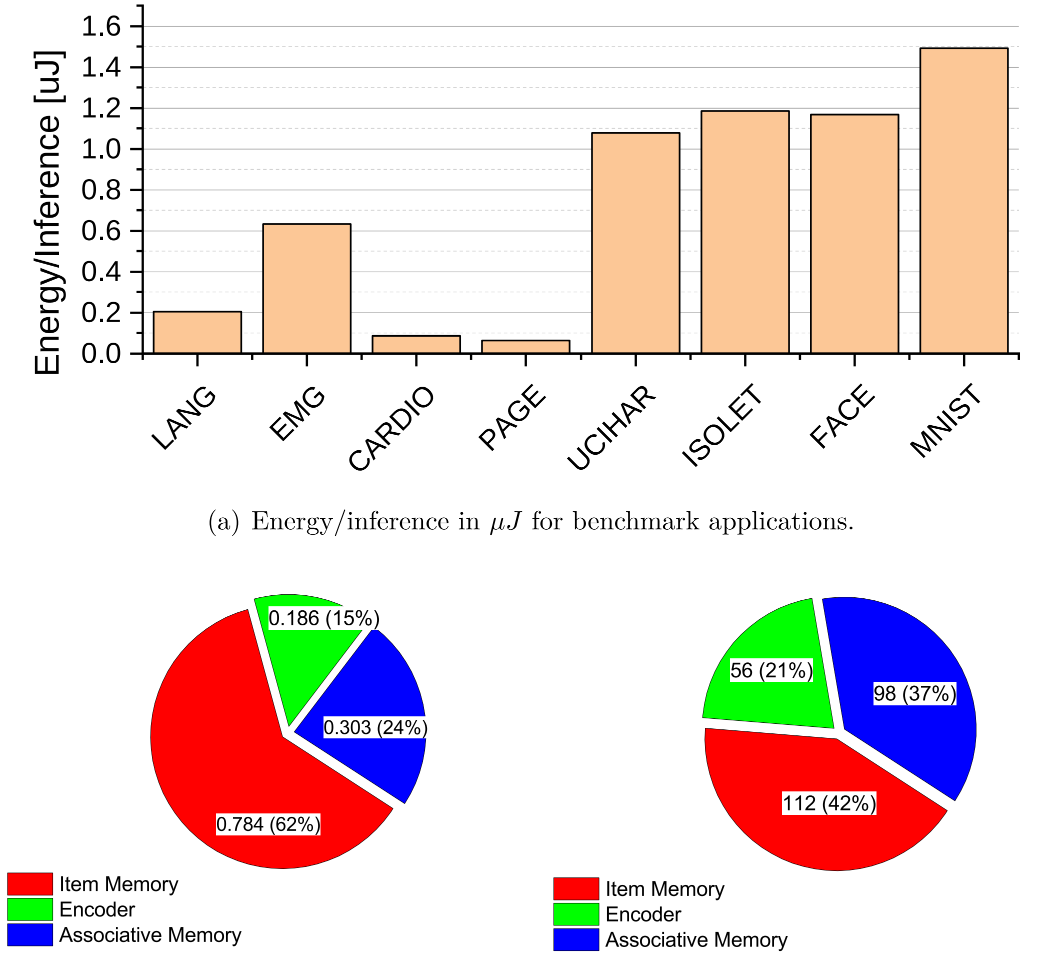
4.3.2 Energy efficiency on a synthesized 28nm processor

A Generic HDC processor was synthesized in a industry-standard 28nm High-K/Metal-gate physical design kit provided by TSMC. The processor consists of an Item Memory with 2048 items including continuous items as shown in figure 4-14, an Associative Memory containing 32 rows and computing hamming distance d_H of all stored vectors with the input in parallel, as shown is fig. 4-16(a). The two-stage Encoder is specified in figure 4-11. Some characteristics of the synthesized data-path is shown in table 4.2.

Property	**Value**
Technology	TSMC 28nm High Performance Mobile
Total Cell Area	1.27 sq. mm.
t_{CLK}	2.4 ns
Total estimated power	267 mW

Table 4.2: Quality of Results (QoR) report for the synthesized Generic processor.

The synthesized data-path is used to simulate the inference computations on the generated net-list, producing logic states at each clock cycle for every wire and standard cell pin. This information is used to estimate power consumption in each clock cycle from energy characterizations from standard cell libraries and capacitances of wires and vias from simple wire models. Such a trace for 20 inferences of Language Recognition is shown in figure 4-20. Using this data, one can calculate the average energy consumed per inference for Language Recognition for these 20 inferences as: total elapsed time × average total power = $(17012 - 7712)$ns × 444mW ≈ 206nJ. Similarly, all applications in the benchmark of section 4.3.1 were profiled for average energy cost per prediction on the synthesized data-path. The results as shown in fig. 4-19 substantiates the claim that a Generic HDC system can be *reasonably* energy efficient – this synthesized data-path requires $\leq 1.5\mu J$ per prediction for the benchmark of section 4.3.1. Hence, a generic HDC chip fabricated and measured in real time is very likely to be energy efficient; it could meet extremely high energy-efficiency requirements necessary for human-centric IoT [149].

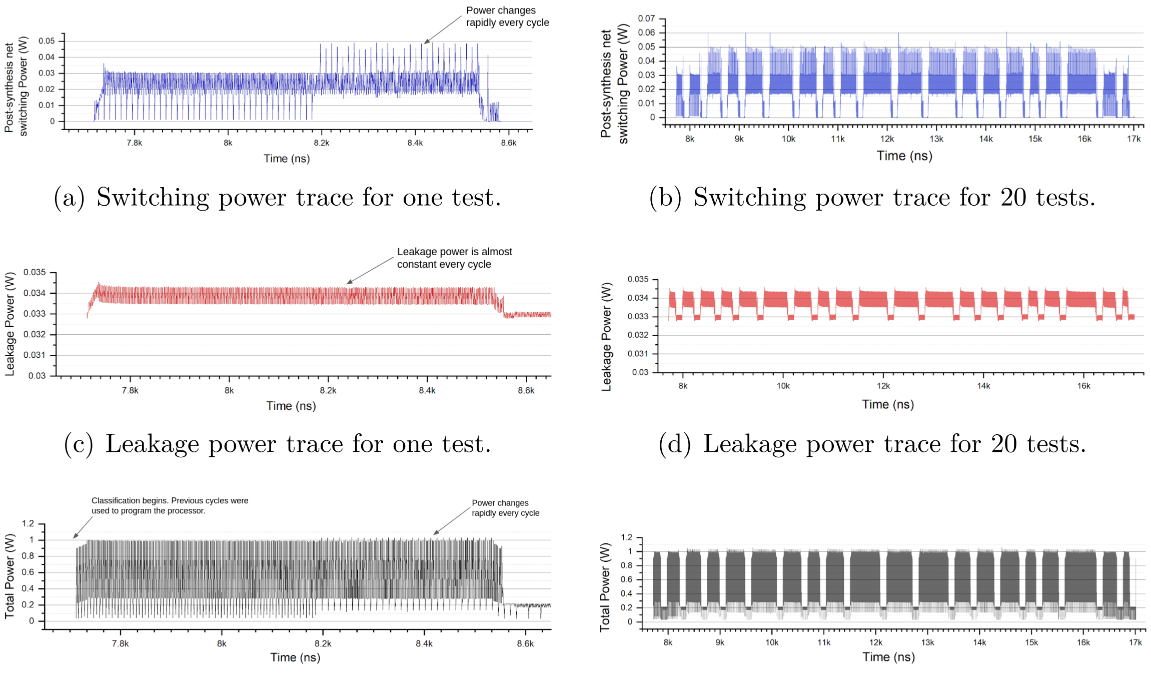


(a) Energy/inference in μJ for benchmark applications.

(b) Total synthesized gates area in mm^2.

(c) Estimated power consumed in mW.

Figure 4-19: Post-synthesis energy per inference for benchmark applications. Detailed descriptions and results are available in [32].



(a) Switching power trace for one test.

(b) Switching power trace for 20 tests.

(c) Leakage power trace for one test.

(d) Leakage power trace for 20 tests.

(e) Total power trace for one test.

(f) Total power trace for 20 tests.

Figure 4-20: Post-synthesis simulation traces for EUROPARL Language Recognition. Using simple parasitic models for wires, via and energy characterization from standard cell libraries, post-synthesis power traces provide useful insights on energy behaviour of the Generic HDC processor. Switching power traces for (a) one inference and (b) 20 inferences use gate and wire capacitances only and should be considered preliminary estimates *correlating* with the actual cost of computation. Leakage power estimates of (b) and (c) are more accurate. The total power trace in (e) clearly shows the encoding phase with a large number of spikes and the associative memory computation with far fewer spikes. Similarly, one could count the encoding phase of each of the 20 inferences from the high-activity durations in (f). The length of the encoding phase in (f) gives a good estimate of the length of the test sentence.

Chapter 5

Architectural techniques for multi-bit Hyper-Dimensional Computing

This chapter considers HDC architectures for applications where more than one bit is used to represent elements of learned hyper-vectors. Multiple bits per vector element are required to accommodate the rapidly growing application space of HDC. Furthermore, multi-bit hyper-vector models almost always have higher performance than their single-bit counterparts. However, multi-bit elements dramatically increase the logical effort of the associative search module in a HDC processor – to the extent that the feasibility of a silicon implementation depends *entirely on the architect's success in limiting this growth in complexity of associative search.*

It is shown that simple, hardware-friendly transformations allow the architect to *fix* the logical cost of associative search for multi-bit hyper-vectors to a small and reasonable *constant* with a bounded loss in accuracy. In other words, given a HDC dimension d, the constant's value does not depend on the encoding expression or application – provided mild conditions on the learned hyper-vectors' probability density are satistisfied. Studies from previous literature which empirically discovered these transformations' effectiveness are reviewed and augmented with complete proofs of the upper-bounded nature of any resultant loss of accuracy. These mathematically rigorous arguments are based on well known results from high-dimensional probability theory and are comprehensively verified by numerical experiments.

An analysis of the proposed transformations' impact on the performance of a HDC application and a study of the reduction in associative search's logical complexity afforded by these transformations concludes this chapter. The key takeaway here is that emerging stochastic and brain-inspired paradigms like HDC allow for *probability-inspired accelerator design.*

5.1 Challenges in multi-bit HDC architectures

Among half a dozen HDC variants [14], the Multiply-Add-Permute (MAP) variant using $0, 1$ or $+1, -1$ as vector elements (called the **binary model** in this chapter) has been most widely adopted in the hardware community due to its hardware-friendliness ([32] provides a brief survey). This is because the binary model with one bit per vector element in the learned hyper-vector is amenable to energy-efficient implementations. However, several advanced algorithms of HDC require storage of partially or completely learned hyper-vectors of non-binary type.

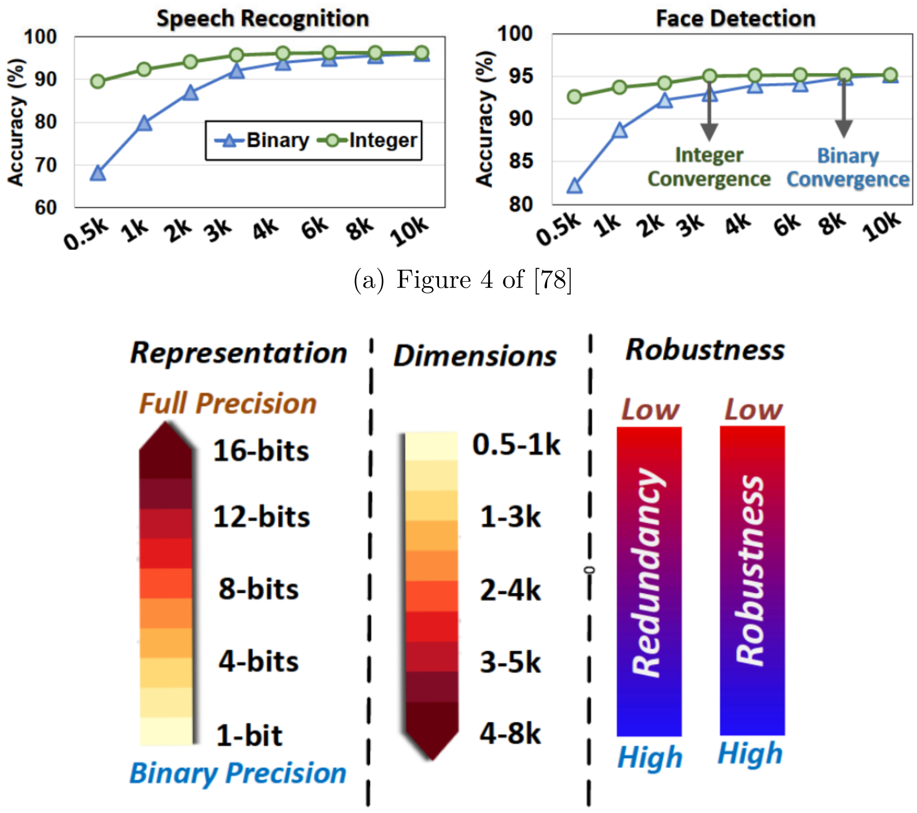
5.1.1 The need for multi-bit HDC

Multiple bits per element of learned hyper-vector is needed due to 2 principal reasons:

1. **Multi-bit learned hyper-vectors lead to higher performance.** The most important reason to design multi-bit HDC architectures is that they almost always have higher performance. When using HDC for supervised classification, as illustrated by fig. 5-1(a), the binary model uniformly has lower classification accuracy than a HDC model using signed integers in two's complement to represent elements of the trained class vector (called **integer model** in this chapter). Furthermore, even though increasing vector dimension improves binary accuracy (see figure 5-1(b)) the integer model uniformly has greater or equal accuracy for every dimension [78].

   This can be explained by the holographic property of HDC: since each element contains identical information as every other element, limiting its representation to one bit limits the total information capacity. This is because each element contains identical information as every other element, limiting the total information capacity [150]. Indeed, [150] concludes that information capacity depends linearly on the vector dimension of the binary model. Contrast this with the integer model, where bits in each element differ in significance, linearly increasing entropy per element; thus requiring a smaller vector dimension for the same information capacity.

2. **Multi-bit Associative Memory helps in re-training.** This is particularly applicable for tasks related to body-sensing and classification, such as EMG hand-gesture recognition [87, 88, 39]. When training a subject's hand gesture, it is necessary to save a partially trained gesture hyper-vector as the subject transitions to rest or another hand-gesture. Thus any natural training procedure, especially one that trains in background without rigid instructions for requiring gestures in a fixed order for fixed durations, necessarily follows a sequence of partial training and re-training (see figure 5-2). Although some re-training procedures exist for the binary model when it already has high accuracy [39], a multi-bit Associative Memory naturally supports re-training – including in cases where the trained vector has poor performance.



(a) Figure 4 of [78]

(b) Figure 5 of [78]

Figure 5-1: Integer models are *uniformly more accurate* than binary models. (a) This is true for all dimensions d. (b) Consequently, much smaller HDC dimension d is required for the integer model to achieve same accuracy as binary model.

> **Perceptron re-training** of the closest known expert model can also improve HDC accuracy [79]. During iteration, if the validation data-point with encoded hyper-vector v of correct class with vector $C_{correct}$ results in a wrong prediction (class with hyper-vector C_{wrong}), we update them as: $C_{wrong} \leftarrow C_{wrong} - v$, $C_{correct} \leftarrow C_{correct} + v$. Clearly, the addition or subtraction of vectors requires storing their elements as integers or real numbers, which necessitates storing learned vectors with multi-bit elements.

In addition to the above reasons, there are several algorithms which are naturally convenient to be expressed on multi-bit HDC models. For example, symbiotic communication using HDC requires the integer model [151]. The factorization algorithm [108, 143] could be performed using both binary and complex unitary vectors, where complex vectors often lead to better empirical performance. Recent work by [142] explores the theoretic development of HDC using vectors from finite fields of a small order, which essentially extends the integer model but guarantees that low precision integers will be sufficient.

Therefore, there is abundant demand from the application space to merit an architectural exploration of multi-bit HDC architectures.

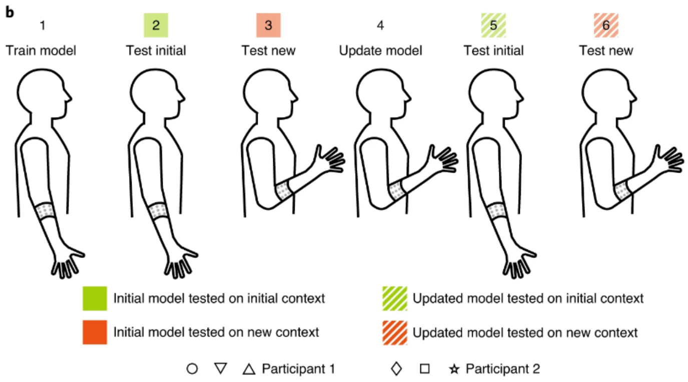

b
1
Train model
2
Test initial
3
Test new
4
Update model
5
Test initial
6
Test new
Initial model tested on initial context
Updated model tested on initial context
Initial model tested on new context
Updated model tested on new context
○ ▽ △ Participant 1
◇ □ ☆ Participant 2

Figure 5-2: Figure 6b of [39]: training, updating hand-gestures and transitioning to other gestures require intermediate storage of learned hyper-vectors.

When a hand gesture is partially trained and a transition to another gesture is initiated, the currently trained hyper-vector needs to be saved in an Associative Memory to allow for inferences using it, and to allow resumption of training that gesture the next time data is available. While bit-mixing [39] to update previously trained vectors is a novel and clever solution to incrementally update hyper-vectors, using the integer model or other multi-bit/element hyper-vector in Associative Memory inherently allows for partial training, intermediate saves and re-training.

5.1.2 Logic complexity of integer Associative Memory

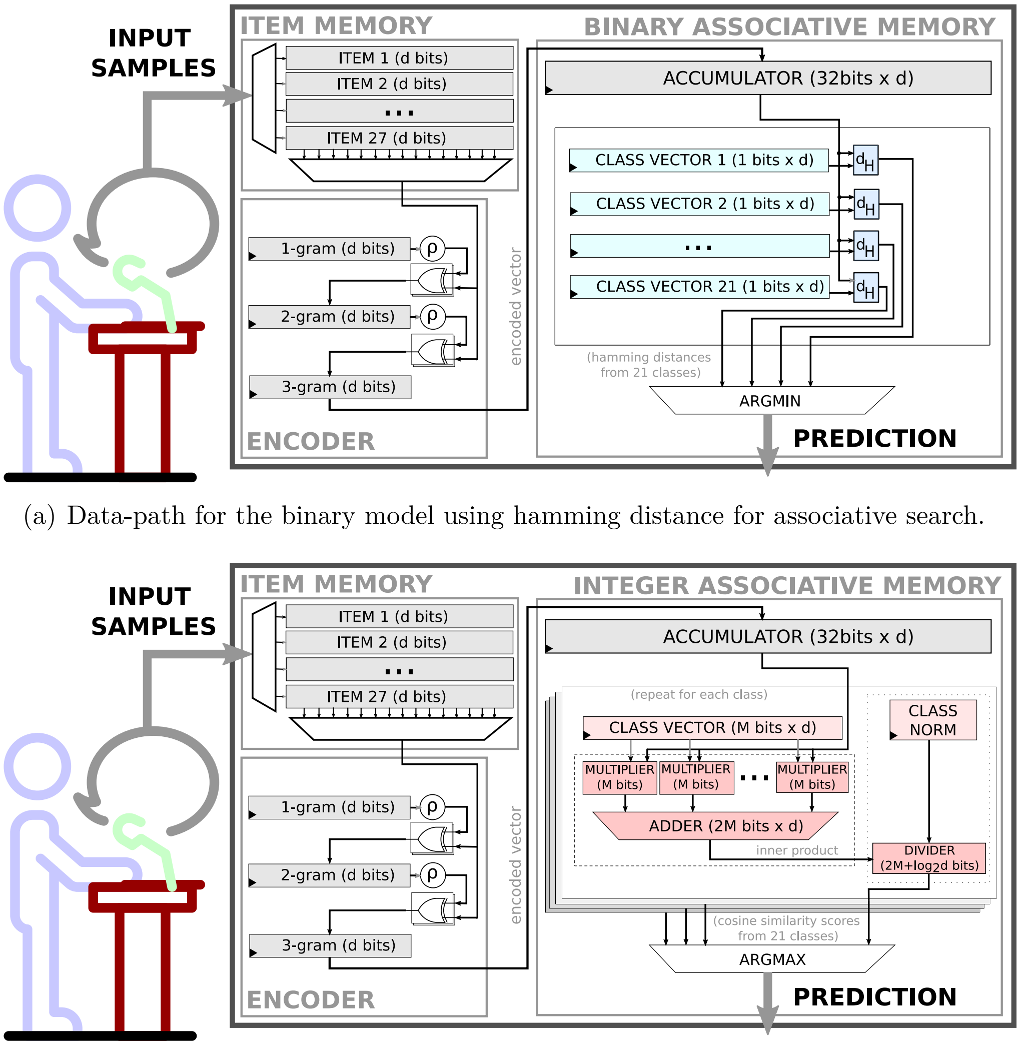
The integer HDC model, the simplest of multi-bit HDC models, is considered for the remainder of this chapter.

Fig. 5-3 shows main components of the HDC data-path for the binary and integer models for performing predictions in a supervised classification task. Within the Encoder, ρ denotes the permutation operation and XOR gates denote the bitwise XOR or binding operation. Since only class vector elements change from binary to integer model, *all hardware modifications to a binary HDC data path needed for the integer model reside within the associative memory.*

The hamming distance $\mathrm{d_H}$ is no longer applicable as a distance metric for the integer model; the cosine similarity $\mathrm{d_{cos}}$, i.e. the inner-product of test vector with each class vector divided by the class vector norm, is applicable and a common choice [37] (also illustrated in figure 5-4(a)).

However $\mathrm{d_{cos}}$ is expensive to implement in hardware (see figure 5-4): M-bit adders in carry-lookahead parallel implementation and Wallace-tree parallel multipliers [152] need $\Theta(M \log M)$ and $\Theta(M^2)$ complex gates respectively with $\Theta(\log M)$ delay [153], and d of them (i.e. thousands) are required for each class vector. A register and divider are needed to store the class norm and divide the inner product for each class. Register memories scale linearly with M. Since there is only one divider per class but thousands of multipliers and adders, its contribution to the total hardware cost is likely negligible. In fact, as shown later in this chapter, the proposed transformations obviate the need for a divider to calculate $\mathrm{d_{cos}}$ altogether.

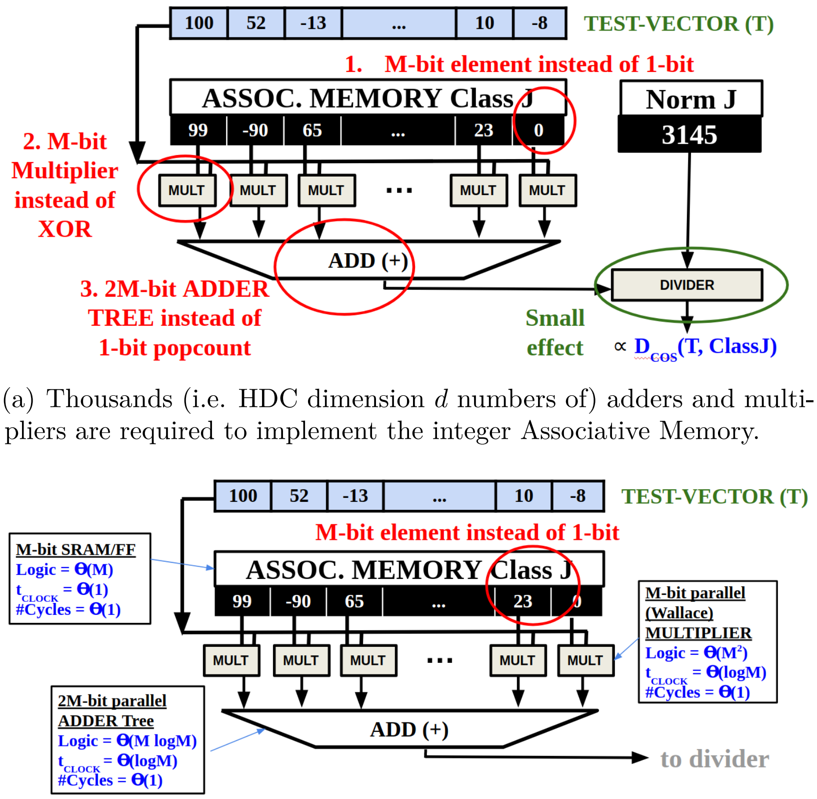
If the number of universal logic gates (called *logical complexity*) is considered as a



(a) Data-path for the binary model using hamming distance for associative search.

(b) Data-path for the integer model using cosine similarity for associative search.

Figure 5-3: Only the associative memories differ among HDC data-paths for the binary and the integer model.
Data-paths for the EUROPARL Language Recognition [65] is fashioned here for (a) the binary model and (b) the integer model. Observe that the entire data-path is identical for both models *except the differing associative memories*. Contrast the adders and multipliers in the integer model's associative memory (b) with the binary model's associative memory (a), where elementary logic gates such as XOR gates and population count are adequate. Any multi-bit Associative Memory like that of integer model in (b) requires higher logical effort due to the inherent complexity of its distance metric computation.



(a) Thousands (i.e. HDC dimension d numbers of) adders and multipliers are required to implement the integer Associative Memory.

(b) The total cost (logic × time taken) of multipliers and adders increases with larger bits/element M. The asymptotic notation $\Theta(\cdot)$ is used here; $\Theta(1)$ indicates that the quantity is constant with respect to M.

Figure 5-4: Costs of integer associative memory grows quicker than linearly with increasing bits/element M.
The logic gates used here are thousands of integer multipliers and adders as shown in (a); the only integer divider, to normalize the inner product with the magnitude of the stored vector, can be ignored when estimating the total hardware effort. (b) lists the time and logical complexity for parallel implementations of vector-store memory, adder and multipliers. The total cost of (parallel) adders and multipliers scale asymptotically as $\Omega(M \log^2 M)$ with increasing bits/element M.

estimate of the adders' and multipliers' power consumption, the product of logical complexity and time taken (composed of critical path delay/clock period and number of cycles) estimates the total energy cost. Therefore, the parallel M-bit adder and multiplier have *energy complexity* of $\Theta(M \log^2 M)$ and $\Theta(M^2 \log M)$ respectively.

The final bits/element M in a trained vector can be arbitrarily large as it is determined by the number of terms superimposed [150], which is usually the number of class examples used in training for supervised application tasks. A large dataset like EUROPARL[90], for the language recognition task [65], would require $M \geq 20$ bits/element of `signed int` for each of the 21 language hyper-vectors. From figure 5-4 and energy complexity relationships, the 20-bit multiplier requires more than 400×

area and 1729× energy of the single-bit multiplier (the AND gate, which is comparable to XOR gate in binary AM), and the 40-bit adder requires more than 213× area and 1133× energy of a half adder (a building block of population count logic). Given that these increases in logic with respect to the binary associative memory are replicated d times per stored hyper-vector where HDC dimension d is in thousands (for example – 21 languages for EUROPARL), any direct implementation of the integer HDC model for a task requiring EUROPARL or similary large dataset is simply not feasible.

Therefore, the only approach to guarantee feasibility of a multi-bit HDC processor is to reduce or (ideally) fix the *required bits/element M* for learned hyper-vectors.

5.1.3 A literature review of multi-bit HDC

Several publications in the HDC literature use non-binary hyper-vectors as their primary data type. For example, [60] studies integer hyper-vectors instead of complex hyper-vectors for modulation and communication over a channel using HDC binding and factorization, [154] compares integer and binary hyper-vectors for robustness to injected errors for supervised classification tasks on a few public data sets, in [72] a Phase-Change Memory (PCM) chip was used to store parts of real-valued hyper-vectors, [155] uses integer hyper-vectors for supervised classifications simulated in an *in-memory* data path, and [156] proposed methods of simulating qubits in quantum circuits using Hyper-Dimensional Computing.

Few published works study the empirical advantages of different strategies to reduce the number of bits per element for non-binary hyper-vectors. In [157] and [85], various quantization strategies are discussed to capture a scalar channel value in multi-channel, multi-modal input streams for classification. Sparse random projections were studied empirically to reduce the memory footprint for integer hyper-vectors in [158] resulting in substantial memory savings. A saturating-counter based superposition scheme was proposed and empirically demonstrated on few supervised classification tasks using public data sets in [76] – as an alternative to thresholding integer hyper-vectors to convert them to binary hyper-vectors.

While published literature provides a wealth of evidence about empirical benefits of various strategies to reduce bits/element in multi-bit HDC models for multiple applications, no analytical investigations to find the mechanism of these improvements exists. Furthermore, it is not clear that the strategies proposed for the specific applications studied will be effective on other applications and data sets. Finally, a theoretical explanation of their usefulness would greatly improve researchers' understanding of the scope of these strategies' advantages and limitations.

The main contributions of this chapter is to augment the multi-bit HDC literature with *mathematically rigorous proofs* establishing their effectiveness. The proven assertions are also empirically verified for an application that agrees with the presumed sufficient conditions. This work was inspired by excellent theoretical treatments of non-binary HDC from the HDC literature – such as [121] for describing real-valued functions represented by hyper-vector-like embeddings and [150] for a capacity analysis of generalized superposition memories.

5.2 Relevant properties of the probability distribution of hyper-vector elements

Notations used in the remainder of this chapter. Hyper-vectors are designated by upper-case letters $X, Y, A, \ldots$ and their elements denoted by corresponding lower-case letters and subscripts $x_i, y_j, a_k, \ldots$. $\mathbf{1}_{\text{condition}}$ is the indicator function for 'condition' (i.e. 1 when 'condition' is true, 0 otherwise). Thus the hyper-vector X is composed of elements $(x_1, x_2, \ldots, x_d)$ where $x_i \in \mathbb{Z}$. Any names of the vectors or their elements can be annotated within parentheses as super-scripts: $x_2^{(\text{4gram})}$ denotes the second element of the hyper-vector X for the 4-gram of inputs. The norm of a vector $\|X\|$ is its euclidean norm $\sqrt{\sum_i x_i^2}$ and $X \cdot Y$ denotes their inner-product $\sum_i x_i y_i$. Random variables are also denoted by upper-case letters $X, U, A, \ldots$ where the context makes it clear if its a random variable or a hyper-vector (or both), and their samples are sub-scripted upper-case letters $X_1, U_i, A_k, \ldots$. $\mathbb{E}[X]$ denotes the expected value of a random variable X and $X \sim Y$ denotes X is identically distributed as Y. If they exist, $f_Z(\cdot)$ denotes the probability density function of random variable Z, μ_X denotes its mean value $\mathbb{E}[X]$ and σ_X denotes its standard deviation i.e. the square-root of variance $\mathbb{E}[(X - \mathbb{E}[X])^2]$. When the random variable X being referred to is clear from the context, μ_X will be abbreviated as μ and σ_X as σ. ■

The integer HDC model is considered for the proposed transformations, where the cosine similarity between hyper-vectors with integer elements $\mathbb{Z}$ is used for the associative search. Following arguments from section 5.1.2, the final objective is to fix bits/element M after a set of hardware-suitable numerical transformations are applied on the stored hyper-vector. Towards that goal, all necessary conditions for the probability distribution of the stored hyper-vector are defined in this section.

Recall that the cosine similarity of two vectors X and Y is given by

$$\mathrm{d}_{\cos}(X, Y) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{k=1}^{d} y_k^2}}$$

For associative search between a common query vector T and stored vectors in the integer model, figure 5-4(a) illustrates the fact that normalization by norm $\|T\| = \sqrt{\sum_{i=1}^{d} t_i^2}$ is not necessary because the query vector is common across all compared cosine similarities. Since cosine similarity calculations are required only for associative search in HDC, the **scaled cosine similarity** score $\overline{\mathrm{d}}_{\cos}(\cdot, \cdot)$ is defined between the stored vector X and common query vector T:

$$\overline{\mathrm{d}}_{\cos}(X, T) \triangleq \frac{X \cdot T}{\|X\|} = \frac{\sum_{i=1}^{d} x_i t_i}{\sqrt{\sum_{i=1}^{d} x_i^2}} \tag{5.1}$$

Now consider how an individual element x_j determines the value of $\overline{\mathrm{d}}_{\cos}(X, T)$. Define the ratio of all other elements and x_j as $c_i \triangleq x_i / x_j$ where $i \neq j$. Then equation

5.1 for scaled cosine similarity can be re-written as:

$$\overline{\mathrm{d}}_{\cos}(X,T) = \frac{\sum_{i=1}^{d} x_i t_i}{\sqrt{\sum_{i=1}^{d} x_i^2}} = \frac{t_j + \sum_{i\neq j} t_i (x_i/x_j)}{\sqrt{1 + \sum_{i\neq j} (x_i/x_j)^2}} = \frac{t_j + \sum_{i\neq j} t_i c_i}{\sqrt{1 + \sum_{i\neq j} c_i^2}} \tag{5.2}$$

If $|x_j|$ is really large in comparison to all other elements of X i.e. $|c_i| \to 0$ for all $i \neq j$, the scaled cosine similarity score from equation 5.2 gives us $\overline{\mathrm{d}}_{\cos}(X,T) \approx \lim_{\forall i\neq j, |c_i|\to 0} \overline{\mathrm{d}}_{\cos}(X,T) = t_j$. Hence, when element x_j has much larger magnitude compared to other elements of X, it alone determines the value of scaled cosine similarity. Conversely, when $|x_j|$ is really small compared to all other elements of X i.e. $|c_i| \to \infty$ for all $i \neq j$, the scaled cosine similarity becomes $\overline{\mathrm{d}}_{\cos}(X,T) \approx \lim_{\forall i\neq j, |c_i|\to \infty} \overline{\mathrm{d}}_{\cos}(X,T) = (\sum_{i\neq j} t_i c_i)/\sqrt{\sum_{i\neq j} c_i^2} = (\sum_{i\neq j} t_i x_i)/\sqrt{\sum_{i\neq j} x_i^2}$. Hence, when element x_j has much smaller magnitude compared to other elements of X, it does not determine the value of scaled cosine similarity.

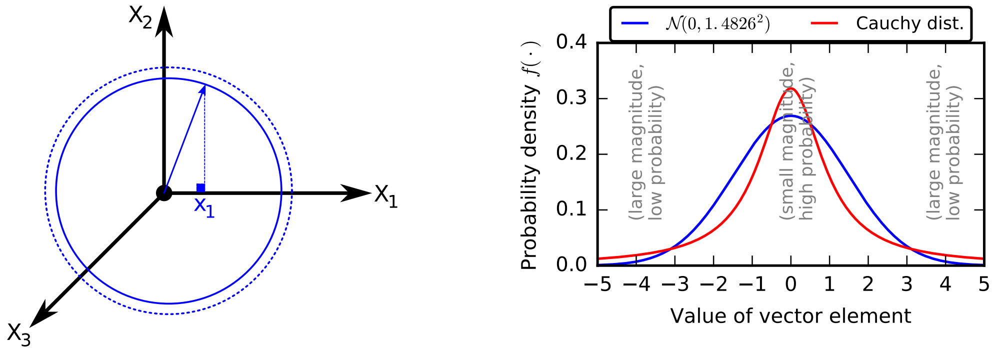
This leads to the first observation required to derive transformations that fix bits/element M of stored hyper-vector X.

Remark 5.2.1. *Only the relatively large-magnitude elements among all elements of the stored vector can meaningfully influence the scaled cosine similarity score.*

5.2.1 Tails of probability density functions

From the holographic property of HDC, recall that encoded hyper-vectors are random variables whose elements are independent and identically distributed. This is because elements of item hyper-vectors are generated independently from the same probability distribution, and the HDC operations multiply, add/superposition or permute do not change the i.i.d. property when encoding hyper-vectors. In other words, all the information in a hyper-vector X is stored in the *probability distribution* of a single element (say, the first element x_1); thousands of elements $x_1, x_2, \ldots, x_d$ for X is required to provide a sufficiently large sample to capture that distribution. The independently and identically distributed nature of hyper-vector elements impose a strong and useful structural constraint on the hyper-vector's d-dimensional probability distribution space: such a distribution has to be **spherically symmetrical** as shown in figure 5-5(a).

Since the influence of an element on the scaled cosine similarity depends on its magnitude *relative to* that of other elements, it is necessary to capture a notion of "spread" or "variation" of the element's distribution. The commonly used metric of variance $\sigma_X^2 = \mathbb{E}[(X-\mu)^2]$ alone is not sufficient to capture this aspect of "variation" in the distribution – the variance can be trivially changed by scaling and there are multiple distributions that share the same variance. Similarly, any of the higher-order central moments $\mathbb{E}[(X-\mu)^k], k \geq 3$ capture an incomplete notion of the distribution's total variation as they too are scale-dependent. Since the *proportion of* large-magnitude elements in the vector is pursued here, a natural next candidate is a dimensionless ratio of central moments. A commonly used ratio of central moments is the **kurtosis** ("tailed-ness" in Greek), defined $\mathbb{E}[(X-\mu)^4]/(\mathbb{E}[(X-\mu)^2])^2$, and is more appropriate



(a) Hyper-vectors belong to spherically-symmetric probability distributions in the d-dimensional space.

(b) Tail characteristics determine proportion of relatively large-magnitude elements in a vector.

Figure 5-5: Properties of hyper-vector probability distribution.
(a) Due to the holographic property of HDC, all hyper-vectors, including items and learned hyper-vectors, have independent and identically distributed elements. Hence the vector's probability distribution is spherically symmetrical i.e. the first coordinate x_1 is identically distributed to all other coordinates on other axes $X_2, X_3, \ldots$.
(b) The proportion of elements with magnitude relatively larger than other elements in a hyper-vector X depends on the tail property of the elements' probability density. Shown density for the Cauchy distribution and the Normal distribution $\mathcal{N}(0, 1.4826^2)$ (i.e. with mean 0 and standard deviation 1.4826) where both distributions have quartiles $-1, 0,$ and 1. Since the Cauchy distribution has heavier tails than the Normal distribution $\mathcal{N}(0, 1.4826^2)$, its hyper-vector will have a higher proportion of elements with relatively larger magnitudes among all its elements.

as it is a (incomplete) metric for the shape of the probability density function f_X.

The Cramér-Chernoff tail bound. A generalization of kurtosis is the Cramér-Chernoff tail bound (see section 2.2 of [159]) for a random variable Z, given by

$$\forall u \geq 0,\ \Pr[Z - \mu \geq u] \leq \exp\Big(\sup_{\lambda \geq 0}(\lambda u - M_{Z-\mu}(\lambda))\Big) \tag{5.3}$$

This characterization of the probability distribution's tail contains all central moments $\mathbb{E}[(Z-\mu)^k], k \geq 2$ as it uses the (central) Moment Generating Function (MGF) for Z defined as $M_{Z-\mu}(\lambda) \triangleq \mathbb{E}[e^{\lambda(Z-\mu)}] = 1 + \sum_{k\geq 2} \frac{\lambda^k}{k!}\mathbb{E}[(Z-\mu)^k]$. For equation 5.3 to be meaningful, the MGF $M_{Z-\mu}(\lambda)$ must be finite in $|\lambda| \leq r$ for some $r > 0$. The Cramér-Chernoff tail inequality is a special case of the general phenomena called **concentration of measures** in high-dimensional geometry. These results were discovered in the late twentieth century and have led to new and profound insights in combinatorics, statistical mechanics and learning theory. Perhaps most importantly, these results allow probabilists and statisticians to refine asymptotic arguments with results requiring only finite sample size, memory and time. The interested reader is referred to the excellent text [159] and monograph [160] for a technical introduction.

The Cramér-Chernoff tail bound can be used to classify probability distributions for a random variable Z admitting a density function $f_Z(\cdot)$ into distinct categories based on the rate of decline in probability density (i.e. the "tail" of the density function) as the quantity $|Z - \mu_Z|$ increases. These results are usually expressed as a univariate function with argument $u \geq 0$ and scalar parameters, which upper bounds the probability $\Pr[|Z - \mu_Z| \geq u]$ (as in equation 5.3). Three categories are well known:

- **Sub-gaussian random variables.** $\Pr[|Z - \mu| \geq u] \leq 2\exp(-u^2/(2\nu))$ for some constant $\nu > 0$ called the variance parameter (see section 2.3 of [159]). The normal distribution $\mathcal{N}(\mu, \sigma^2)$ (i.e with mean μ and variance $\sigma^2 > 0$) is a sub-gaussian random variable with variance parameter $\nu = \sigma^2$. Sub-gaussian random variables have one of the steepest decline in probability density with increasing $|Z - \mu|$ – they are a family of distributions with *light tails*. In practise, sub-gaussian random variables are often well-approximated by the Normal distribution with a fitted mean μ and variance $\sigma^2 = \nu$.
- **Sub-exponential random variables.** Using Bernstein's characterization (see proposition 2.10 of [160]), sub-exponential random variables satisfy the tail inequality $\Pr[|Z - \mu| \geq u] \leq 2\exp\Big(-u^2/(2\nu + 2bu)\Big)$ where variance parameter $\nu > 0$ and scale parameter $b \geq 0$. The sub-exponential's rate of decline in density is slower than that of sub-gaussian random variables (with the same ν) whenever $b > 0$; when $b = 0$ the sub-exponential random variable becomes sub-gaussian. The Chi-squared distribution with d degrees of freedom is a prominent example of sub-exponential random variable and its concentration of probability density is the central result used later in this chapter: from Lemma 5.2.1, χ_d^2 is sub-exponential with parameters $\nu = 2\sqrt{d}$ and $b = 4$.
- **Sub-gamma random variables.** $\Pr[|Z - \mu| \geq u] \leq 2\exp\Big(-\frac{(\nu+cu)-\sqrt{\nu^2+2\nu cu}}{c^2}\Big)$ for variance parameter $\nu > 0$ and scale parameter $c \geq 0$ (see Theorem 2.3 of [159]). Clearly, the sub-gamma random variable has a rate of density decline slower than sub-exponential and sub-gaussian random variables – they are a family of distributions with *heavy tails*. The Gamma distribution is an example of sub-gamma random variable. Finally, note that when $c = 0$ sub-gamma random variables become sub-gaussian with parameter ν, since $\lim_{c\to 0} 2\exp\Big(\frac{\sqrt{\nu^2+2\nu cu}-(\nu+cu)}{c^2}\Big) = \lim_{c\to 0} 2\exp\Big(\frac{\nu\sqrt{1+(2cu/\nu)}-(\nu+cu)}{c^2}\Big) = \lim_{c\to 0} 2\exp\Big(\frac{\nu(1+(1/2)(2cu/\nu)-(1/8)(2cu/\nu)^2)-(\nu+cu)}{c^2}\Big) = 2\exp(-u^2/(2\nu))$. Therefore, it may be possible to approximate a sub-gamma random variable with a Normal random variable with a fitted mean μ and variance $\sigma^2 = \nu$ if the scale parameter c is very small compared to $\sqrt{\nu}$.

In summary, the nature of the Cramér-Chernoff tail bound provides a complete characterization of the elements' probability density function which is necessary to design transformations that exploit remark 5.2.1. Figure 5-5(b) illustrates this for the Cauchy distribution (with extremely heavy tails – its mean, variance and higher-order central moments do not exist) and a comparable Normal distribution: a heavier tail of the density indicates a larger proportion of elements influencing the cosine similarity.

5.2.2 Chi-squared concentration

Following the discussion of tail bounds of probability density functions, the main result of this chapter pertaining to the tail behavior of the Chi-squared random variable and its relation to the Normal distribution $\mathcal{N}(\mu, \sigma^2)$ is stated.

Definition 5.2.1 (χ^2_d random variable)**.** Given independent $x_1, x_2, \ldots, x_d \sim N(0,1)$, the squared euclidean norm of the vector $X = (x_1, x_2, \ldots, x_d)$ given by $\|X\|^2 = \sum_{i=1}^{d} x_i^2$ obeys the Chi-squared distribution with d-degrees, denoted χ^2_d. Conversely, if $Z \sim \chi^2_d$ then Z can be written as $\sum_{i=1}^{d} x_i^2$ where $x_1, x_2, \ldots x_d$ are i.i.d. random variables obeying $\mathcal{N}(0,1)$. Its expected value is $\mathbb{E}[Z] = d$.

The following lemma 5.2.1 is necessary to prove the concentration of density result for χ^2_d in lemma 5.2.2. Short proofs adapted from example 2.8 of [160] are provided for both these lemmas. Lemma 5.2.2 is weaker than well known results such as described in (lemma 1 of [161]). However, it is sufficiently strong for our purpose.

Lemma 5.2.1. *If $Z \sim \chi^2_d$ then the central Moment Generating Function (MGF) satisfies $M_{Z-\mu_Z}(u) = \mathbb{E}[e^{u(Z-d)}] \leq e^{2du^2}$ for $|u| < \frac{1}{4}$.*

Proof. Begin from $Z = \sum_i x_i^2$ and simplify:

$$\begin{aligned}\mathbb{E}[e^{u(Z-d)}] &= \prod_{i=1}^{d} \mathbb{E}[e^{u(x_i^2-1)}]\ (x_i \text{ are independent})\\ &= (2\pi)^{-\frac{d}{2}} \prod_{i=1}^{d} \int_{-\infty}^{+\infty} e^{u(x_i^2-1)} e^{-x_i^2/2} dx_i = (2\pi)^{-\frac{d}{2}} e^{-du} \prod_{i=1}^{d} \int_{-\infty}^{+\infty} e^{-(1-2u)x_i^2/2} dx_i\\ &= (2\pi)^{-\frac{d}{2}} e^{-du} \prod_{i=1}^{d} \sqrt{\frac{2\pi}{1-2u}}\ (\text{iff } u < 1/2) = e^{-du}(1-2u)^{-\frac{d}{2}} = (e^{-u}(1-2u)^{-1/2})^d\end{aligned}$$

The MGF exists iff $u < \frac{1}{2}$. Use $e^{-u}(1-2u)^{-1/2} < e^{2u^2}$ for $|u| < \frac{1}{4}$ to get the result. ■

Lemma 5.2.2 (χ^2_d concentration)**.** *If $Z \sim \chi^2_d$ then $\Pr[\frac{Z}{d} - 1 \geq t] \leq e^{-dt^2/8}$ and $\Pr[\frac{Z}{d} - 1 \leq -t] \leq e^{-dt^2/8}$ whenever $0 < t < 1$.*

Proof. Recall Markov's inequality: for non-negative random variable $X \geq 0$ and all $t > 0$, $\Pr[X \geq t] \leq \mathbb{E}[X]/t$. Now if $\lambda \geq 0$, $e^{\lambda x}$ is non-decreasing function of x. Thus, for all $t, \lambda \geq 0$, $\Pr[Z - d \geq dt] \leq \Pr[e^{\lambda(Z-d)} \geq e^{\lambda dt}] \leq e^{-\lambda dt}\,\mathbb{E}[e^{\lambda(Z-d)}]$ using Markov's on $e^{\lambda(Z-d)}$. Use Lemma 1 to get $\Pr[Z - d \geq dt] \leq e^{-\lambda dt + 2d\lambda^2}$ for $0 \leq \lambda < 1/4$. The expression $e^{-\lambda dt + 2d\lambda^2}$ has as global minimum at $\lambda^* = t/4$ which falls in range $0 \leq \lambda^* < 1/4$ whenever $0 < t < 1$. Thus $\Pr[Z - d \geq dt] \leq e^{-\lambda^* dt + 2d(\lambda^*)^2} = e^{-dt^2/8}$.

The argument for $\Pr[Z - d \leq -dt]$ is analogous: if $\lambda \leq 0$, $e^{\lambda x}$ is non-increasing function of x. Thus, for all $t, \lambda \leq 0$, $\Pr[Z - d \leq -dt] \leq \Pr[e^{\lambda(Z-d)} \geq e^{-\lambda dt}]$. Using Lemma 1 and Markov's, $\Pr[Z - d \leq -dt] \leq e^{\lambda dt + 2d\lambda^2}$ with global minimum at $\lambda^* = -t/4$. Substitution gives the result. ■

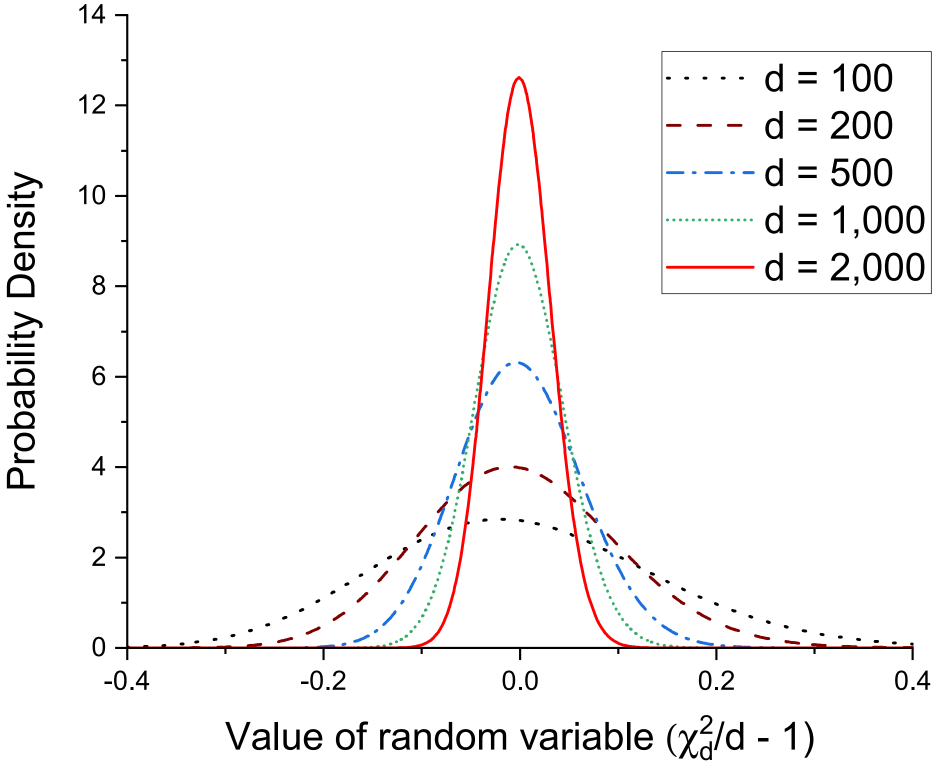


Figure 5-6: Concentration of probability density around d for χ_d^2.
Note that as HDC dimension d increases, χ_d^2/d is very likely to be close to 1.

Lemma 5.2.2 shows that the Chi-squared density falls as $e^{-\Theta(|\chi_d^2-d|^2)}$. Given that the distribution of hyper-vectors X is spherically symmetric (as shown in figure 5-5(a)), the essential identity of the learned vector is the distribution of its norm $\|X\|$. If the hyper-vectors X are normally distributed, lemma 5.2.2 determines that – because of its high dimensional nature – the majority of the probability density for $\|X\|^2$ lies within a small interval around its expected value $\mathbb{E}[\|X\|^2]$. This fact has been widely employed for uniform sampling from high-dimensional spheres using standard normal variables (for example, see items 26 and 27 by Eugene Salamin in HAKMEM [162]).

The next sub-section discusses the EUROPARL dataset and the normal distribution as an approximation of its language hyper-vectors' elements.

5.2.3 Normality assumption and the EUROPARL dataset

Informally, if it can be shown that transformations exist which can reduce the elements' magnitude (therefore, bits/element M) of hyper-vectors *without significantly altering* the vector's radial distribution (i.e. of $\|X\|$ or $\|X\|^2$), one can prove that the resulting inaccuracy in cosine similarity will be negligible. Therefore, from section 5.2.2 if the hyper-vector elements happen to be independently and Normally distributed, one way to proceed is to show that the concentration of its radial χ_d^2 distribution remains unchanged by applying the proposed transformations. Note that the approach of using the density concentration of the radial distribution *as a consequence* of an assumption on each element's distribution simplifies the problem greatly – for one now need only consider *element-wise transformations* which are easily implementable in digital logic.

Hence, the Normality assumption on elements of hyper-vectors stored in the Associative Memory is defined below. This assumption is necessary to produce concrete mathematical arguments supporting the efficacy of the proposed transformations. The results derived in this chapter can be easily extended to other distribution families of Cramér-Chernoff tail behavior; chapters 2 of [159, 160] are recommended starting points.

Definition 5.2.2 (Normality assumption)**.** The elements of a hyper-vector stored in the associative memory are independent and identically distributed and follow a (zero mean) Normal distribution.

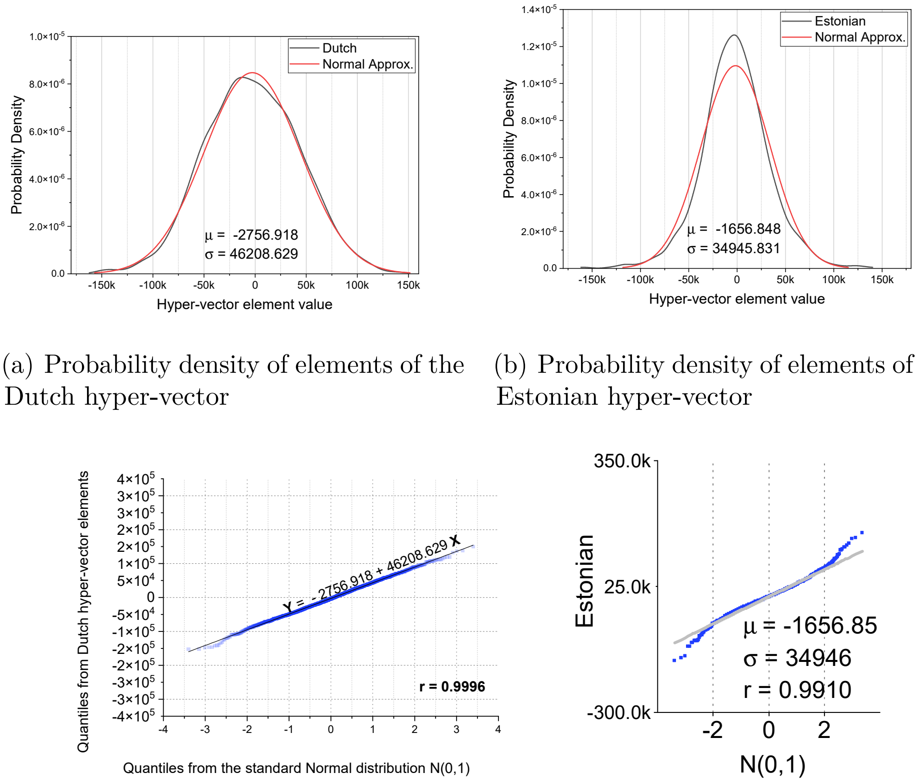
Shannon's seminal work [163] argues that for classification, languages can be adequately modeled as *independent* samples from its empirical n-gram distribution. Independence is essential because when HDC is used for language recognition [65], the language hyper-vector is formed by superposition of bipolar 4-gram vectors:

$$X^{(\text{language})} = A^{(\text{first 4gram})} + B^{(\text{second 4gram})} + \dots$$

Since $A^{(\text{first 4gram})}, B^{(\text{second 4gram})}, \dots$ are bipolar vectors which are *independent* as the 4-gram samples are independent, using de Moivre's theorem [164] the sum of independent bipolar random variables may be approximated as a normally-distributed random variable. And since the elements of bipolar vectors have a mean value of 0, the final elements of the superimposed language hyper-vector can be accurately approximated as a zero-mean normal distribution.

The EUROPARL dataset is a good dataset to verify proposed transformations as it is a really large. Furthermore, when used for language recognition [65], the normality assumption in definition 5.2.2 is valid for EUROPARL. The Dutch language hyper-vector agrees the most with Normality assumption giving a coefficient of determination $r = 0.9996$ (see figure 5-7(a) and (c)). The Estonian language hyper-vector is least agreeable to the Normality assumption giving the smallest coefficient of determination $r = 0.9910$ among all european languages in EUROPARL (see figure 5-7(b) and (d)). Fig. 5-8 shows the quantile-quantile plot for the empirical distribution of elements versus the fitted normal distribution.

Note that for at least 95% elements of all language hyper-vectors – i.e. $\mathcal{N}(0, 1)$ is within 2 standard deviations of mean 0 – a straight line is a great fit in the quantile-



(a) Probability density of elements of the Dutch hyper-vector

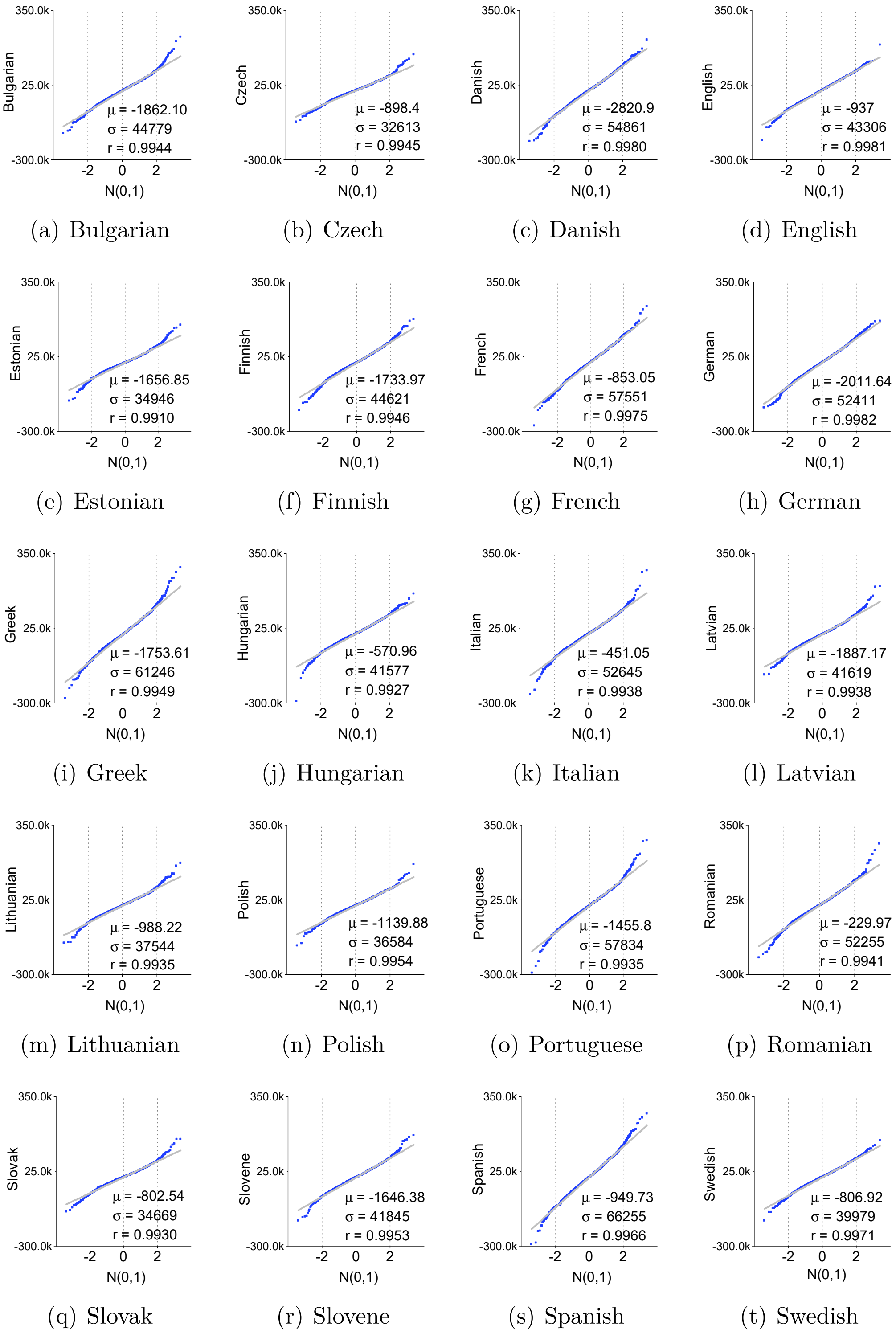
(b) Probability density of elements of the Estonian hyper-vector

(c) Quantile-quantile plot for elements of Dutch hyper-vector and fitted normal distribution.

(d) Quantile-quantile plot for elements of Estonian hyper-vector and fitted normal distribution.

Figure 5-7: Normality assumption for best fitting (Dutch) and least fitting (Estonian) language hyper-vectors in the EUROPARL corpus.

quantile plot of figure 5-8. Since $\mu/\sigma < 0.06$ for the fitted normal distribution of all language hyper-vectors, the mean value is essentially zero. Therefore, there is little evidence to reject the normality assumption in definition 5.2.2 for EUROPARL.



(a) Bulgarian (b) Czech (c) Danish (d) English

(e) Estonian (f) Finnish (g) French (h) German

(i) Greek (j) Hungarian (k) Italian (l) Latvian

(m) Lithuanian (n) Polish (o) Portuguese (p) Romanian

(q) Slovak (r) Slovene (s) Spanish (t) Swedish

Figure 5-8: Normality assumption for all except Dutch in the EUROPARL corpus. Quantile-quantile plots for the empirical distribution of language hyper-vectors' elements ($d = 2048$) and the best-fitting normal distribution are shown. Elements' distribution are modeled as $Y = \mu + \sigma X$ where $X \sim \mathcal{N}(0, 1)$. Coefficients of determination $0 \leq r \leq 1$ are also specified; $r = 1$ indicates perfect modeling.

5.3 Transformations for precision reduction

Before applying numerical transformations, encoded hyper-vectors must be held in `ACCUMULATOR` (fig. 5-3). To allow training large datasets, `ACCUMULATOR` must have large enough bits/element (eg. 32 bits/element as shown in figure 5-3) for both binary and multi-bit HDC data paths. In other words, the HDC processor must hold the encoded hyper-vector *once* in the `ACCUMULATOR` with its full precision before it can be transformed, thereby reducing the bits/element M for cosine distance calculation and Associative Memory vector storage as illustrated in figure 5-4 (b).

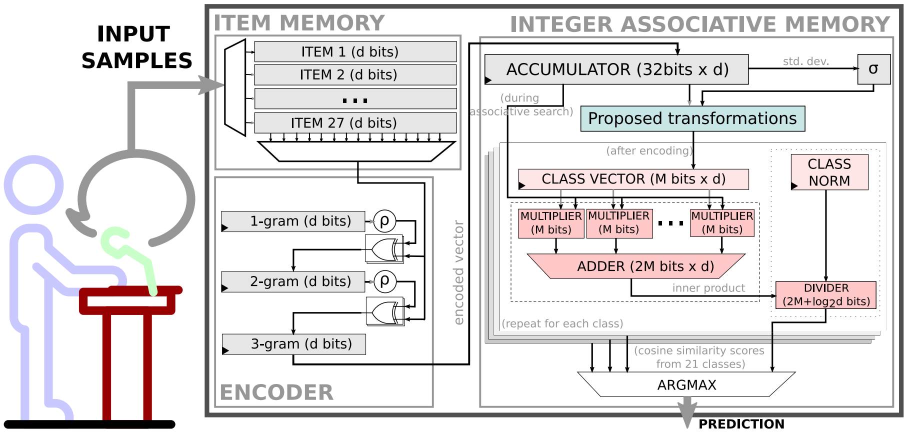


Figure 5-9: The proposed transformations require elements' standard deviation σ. After encoded hyper-vectors are available in the `ACCUMULATOR`, the standard deviation of vector elements σ is calculated prior to applying the transformations.

The proposed transformations require prior calculation of the standard deviation across elements of the encoded hyper-vector (see figure 5-9). Additional logic is required to calculate the standard deviation across elements, which is very close to that of the fitted normal distribution from figure 5-8. However, this calculation is done only once: after encoding and before the numerical transformations. Hence, these transformations may be considered as effectively *normalization* of the encoded hyper-vectors.

It is assumed that $d = 2048$ since this is sufficient for most applications [32].

5.3.1 Saturation

The first objective is to limit the maximum magnitude of the encoded hyper-vector elements. The idea is to use χ_d^2 concentration to show that it is very unlikely for *any element* of the encoded hyper-vector to have a magnitude higher than a large multiple of standard deviation σ.

The **saturation** of hyper-vector elements x_i by a scalar $S > 0$ is defined as:

$$\text{Saturate}(x_i; S > 0) \triangleq \begin{cases} +S, & \text{if } x_i \geq S \\ x_i, & \text{if } -S \leq x_i \leq S \\ -S, & \text{if } x_i \leq -S \end{cases}$$

The element-wise saturation can be readily implemented using digital comparators.

One can now use the normality assumption of encoded hyper-vector elements and χ_d^2 concentration to derive the expression for S in terms of the probability that no hyper-vector element is larger than S.

Corollary 5.3.0.1. *Let $X = (x_1, x_2, \ldots, x_d)$ where $x_i \sim \mathcal{N}(0, \sigma^2)$ are independent. Given $S > 0$, define $Y \triangleq \text{Saturate}(X; S)$ to be the hyper-vector obtained from element-wise saturation of X by S. If $e^{-d/8} < \delta < 1$ then $S \geq \sigma\sqrt{d + \sqrt{8d \log 1/\delta}}$ implies* $\Pr[Y = X] \geq 1 - \delta$. *(Note that the condition $e^{-d/8} < \delta < 1$ is not really constraining δ: $d \geq 500 \implies e^{-d/8} < 7.2 \times 10^{-28}$.)*

Proof. See that $\max_{1 \leq i \leq d} |x_i| \leq \|X\|$. Thus, $\Pr[Y = X] = \Pr[\max_i |x_i| \leq C] \geq \Pr[\|X\| \leq C]$. Since $x_i \sim \mathcal{N}(0, \sigma^2)$, we have $\|X\|^2/\sigma^2 \sim \chi_d^2$. From Lemma 5.2.2, $\Pr[\|X\|^2/\sigma^2 \geq d(1 + t)] \leq e^{-dt^2/8}$ whenever $0 < t < 1$. Substituting t from $e^{-dt^2/8} \leq \delta$ gives $d(1 + t) \geq d + \sqrt{8d \log 1/\delta}$ which is valid provided $\delta > e^{-d/8}$. Hence we have $\Pr\left[\|X\| \leq \sigma\sqrt{d + \sqrt{8d \log 1/\delta}}\right] \geq 1 - \delta$. Putting it all together, if $S \geq \sigma\sqrt{d + \sqrt{8d \log 1/\delta}}$ then $\Pr[Y = X] = \Pr[\max_i |x_i| \leq S] \geq \Pr\left[\|X\| \leq S\right] \geq 1 - \delta$ ■

Saturation has been studied in [150] – defined there as *clipping* – in the context of memory capacity of neural networks. It has also been noted in [165] for its empirical effectiveness in dimensionality reduction of hyper-vectors.

From the corollary above, there is at least 99% chance of *no change* when saturating at $S \geq 1.05 \times \sigma\sqrt{2048} \approx 48\sigma$ when HDC dimension is $d = 2048$. An advantage of using χ_d^2 concentration rather than error function $\text{erf}(\cdot)$ for deriving bounds for S is that while the former readily produces precise bounds for the entire hyper-vector, the latter's expression $(\text{erf}(24\sqrt{2}))^{2048}$ is harder to calculate as $\text{erf}(24\sqrt{2})$ is vanishingly small.

As demonstrated in figure 5-10, for $S = 48\sigma$ and HDC dimension $d = 2048$ all EUROPARL language hyper-vectors remain unchanged. Furthermore, as S decreases to 0 the testing accuracy decreases as more hyper-vector elements get saturated.

While saturation limits the magnitude of hyper-vector elements to a large multiple of σ, it is still dependent on the standard deviation given by $\sigma = \sqrt{\frac{\sum_{i=1}^{d}(x_i - \mu)^2}{d-1}} \approx$

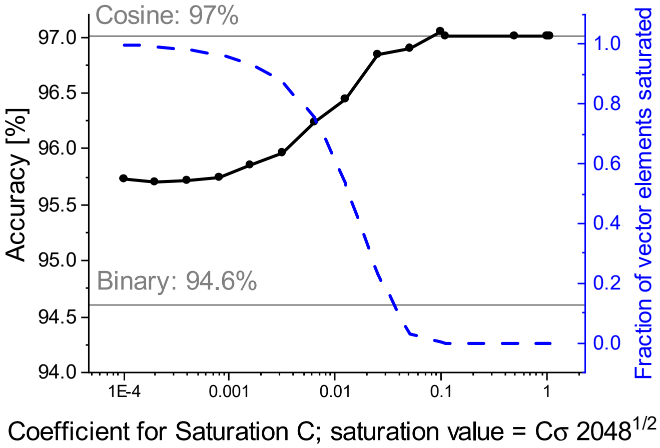


Figure 5-10: Accuracy of saturated language hyper-vectors for EUROPARL. The right vertical axis used by the dashed plot shows the fraction of hyper-vector elements saturated when $S = C\sigma\sqrt{d}$.

$\sqrt{\frac{\sum_{i=1}^{d} x_i^2}{d-1}} \propto \|X\|$ since the mean of hyper-vector elements $\mu = \sum_{i=1}^{d} x_i \approx 0$. The next transformation removes this dependence on $\|X\|$.

5.3.2 Thresholding

The **thresholding** of hyper-vector elements x_i by a scalar $T > 0$ is defined as:

$$\text{Threshold}(x_i; T > 0) \triangleq \begin{cases} 0 & \text{if} - T \leq x_i \leq T \\ x_i & \text{if otherwise} \end{cases}$$

The idea behind thresholding is that when hyper-vector elements of small magnitude are zeroed out, the error caused in the cosine distance $\mathrm{d_{cos}}$ with *any other vector* can be upper bounded by a small number. Thresholding has been proposed [117] for increased sparsity of hyper-vectors in supervised classification tasks. A similar idea of dropping vector dimensions with high inter-vector variance in the associative memory has also been proposed in [166]. It can be implemented by a digital comparator for each element of the hyper-vector.

Theorem 5.3.1 (Thresholding)**.** *Let $X = (x_1, x_2, \ldots, x_d)$ where $x_i \sim \mathcal{N}(0, \sigma^2)$ are independent. Given $C > 0$, let $\tau_C(X) = Threshold(X; C > 0)$ be the hyper-vector obtained by applying thresholding function element-wise on X. Then for any vector $u \in \mathbb{R}^d$, error $0 < \epsilon \leq 1$ and $e^{-d/8} < \delta < 1$, we have* $\Pr[|d_{cos}(u, X) - d_{cos}(u, \tau_C(X))| \leq \epsilon] \geq 1 - \delta$ *whenever* $C \leq \frac{\sigma}{2}\epsilon^2\sqrt{1 - \sqrt{\frac{8}{d}\log 1/\delta}}$

Proof. From lemma 5.2.2, $\Pr\left[\|X\| \geq \sigma\sqrt{d - \sqrt{8d\log 1/\delta}}\right] \geq 1 - \delta$ whenever $e^{-d/8} <$

$\delta < 1$. Then the error in cosine similarity due to saturation can be simplified as:

$$
\begin{aligned}
|\,\mathrm{d}_{\cos}(u, X) - \mathrm{d}_{\cos}(u, \tau_C(X))| &= \Big|\frac{u}{\|u\|} \cdot \frac{X}{\|X\|} - \frac{u}{\|u\|} \cdot \frac{\tau_C(X)}{\|\tau_C(X)\|}\Big| \\
= \Big|\frac{u}{\|u\|} \cdot \Big(\frac{X}{\|X\|} - \frac{\tau_C(X)}{\|\tau_C(X)\|}\Big)\Big| &\leq \Big\|\frac{X}{\|X\|} - \frac{\tau_C(X)}{\|\tau_C(X)\|}\Big\| \text{(Cauchy-Schwarz inequality)} \\
= \sqrt{2\Big(1 - \frac{X \cdot \tau_C(X)}{\|X\|\|\tau_C(X)\|}\Big)} &= \sqrt{2\Big(1 - \frac{\sum_{i=1}^{d} x_i^2 \mathbf{1}_{|x_i| \geq C}}{\|X\|\|\tau_C(X)\|}\Big)} \text{ (definition of } \tau_C(\cdot)) \\
&= \sqrt{2\Big(1 - \frac{\|\tau_C(X)\|^2}{\|X\|\|\tau_C(X)\|}\Big)} \text{ (expression for } \|\tau_C(X)\|^2) \\
&= \sqrt{2\frac{\|X\| - \|\tau_C(X)\|}{\|X\|}} \leq \sqrt{2\frac{\|X - \tau_C(X)\|}{\|X\|}} \text{ } (\triangle \text{ inequality}) \\
&= \sqrt{2\frac{\sqrt{\sum_{i=1}^{d} x_i^2 \mathbf{1}_{|x_i| < C}}}{\|X\|}} < \sqrt{2\frac{\sqrt{\sum_{i=1}^{d} C^2}}{\|X\|}} < \sqrt{2C\frac{\sqrt{d}}{\|X\|}} \\
&\leq \sqrt{\frac{2C}{\sigma\sqrt{1 - \sqrt{(8/d)\log 1/\delta}}}} \text{ with probability} \geq 1 - \delta
\end{aligned}
$$

Using the condition $C \leq \frac{\sigma}{2}\epsilon^2\sqrt{1 - \sqrt{(8/d)\log 1/\delta}}$ gives the result. ■

The key observation from Theorem 5.3.1 is that the magnitude of threshold C is also proportional to the standard deviation σ of hyper-vector elements. Thus, following the results of corollary 5.3.0.1 for saturation, one may threshold the saturated hyper-vector with T from Theorem 5.3.1.

The thresholded hyper-vector may be divided by the threshold T element-wise without affecting the cosine similarity as it is norm invariant. Given that both S and T are proportional to σ, the largest magnitude of the saturated and thresholded hyper-vector divided element-wise by T is S/T, which is *independent of* σ. This is shown in the following corollary.

Corollary 5.3.1.1 (Saturation, Thresholding and integer divison by threshold)**.** *Let $X = (x_1, x_2, \ldots, x_d)$ where $x_i \sim \mathcal{N}(0, \sigma^2)$ are independent. Given $C > 0$, let $\tau_C(X) =$ Threshold$(X; C > 0)$ be the hyper-vector obtained by applying thresholding function element-wise on X. Given $0 < \epsilon \leq 1$ and $2e^{-d/8} < \delta < 1$ and any vector $u \in \mathbb{R}^d$, let $a = \sigma\sqrt{d + \sqrt{8d\log(2/\delta)}}$ and $b = \frac{\sigma}{2}\epsilon^2\sqrt{1 - \sqrt{(8/d)\log(2/\delta)}}$. Then the elements y_i of hyper-vector $Y = \tau_b(Z)/b$, where $Z =$ Saturate$(X; a)$, satisfies $\max_{1 \leq i \leq d} |y_i| \leq 2\frac{\sqrt{d}}{\epsilon^2}\sqrt{\frac{1+\sqrt{(8/d)\log(2/\delta)}}{1-\sqrt{(8/d)\log(2/\delta)}}}$ and for all $u \in \mathbb{R}^d$, $\Pr[|\,\mathrm{d}_{cos}(u, X) - \mathrm{d}_{cos}(u, Y)| \leq \epsilon] > 1 - \delta$*

Proof. For the first part, note that the maximum absolute value of elements of $Z = \mathrm{Saturate}(X; a)$ is a, which remains unchanged by the subsequent thresholding $\tau_b(\cdot)$. Thus, the maximum absolute value of $Y = \tau_b(Z)/b$ is a/b.

For the second part, begin from $\mathrm{d_{cos}}(u, Y) = \mathrm{d_{cos}}(u, bY)$ and simplify:

$$\begin{aligned}
&\Pr[|\,\mathrm{d_{cos}}(u, X) - \mathrm{d_{cos}}(u, Y)| \leq \epsilon] \\
&= \Pr[|\,\mathrm{d_{cos}}(u, X) - \mathrm{d_{cos}}(u, \tau_b(Z))| \leq \epsilon] \\
&= \Pr[|\,\mathrm{d_{cos}}(u, X) - \mathrm{d_{cos}}(u, \tau_b(Z))| \leq \epsilon | Z = X] \Pr[Z = X] \\
&+ \Pr[|\,\mathrm{d_{cos}}(u, X) - \mathrm{d_{cos}}(u, \tau_b(Z))| \leq \epsilon | Z \neq X] \Pr[Z \neq X] \\
&\geq \Pr[|\,\mathrm{d_{cos}}(u, X) - \mathrm{d_{cos}}(u, \tau_b(Z))| \leq \epsilon | Z = X] \Pr[Z = X] \\
&\geq \Pr[|\,\mathrm{d_{cos}}(u, X) - \mathrm{d_{cos}}(u, \tau_b(X))| \leq \epsilon](1 - \delta/2) \text{ (from Corollary 5.3.0.1)} \\
&\geq (1 - \delta/2)^2 \text{ (from Theorem 5.3.1)} \\
&> 1 - \delta
\end{aligned}$$

■

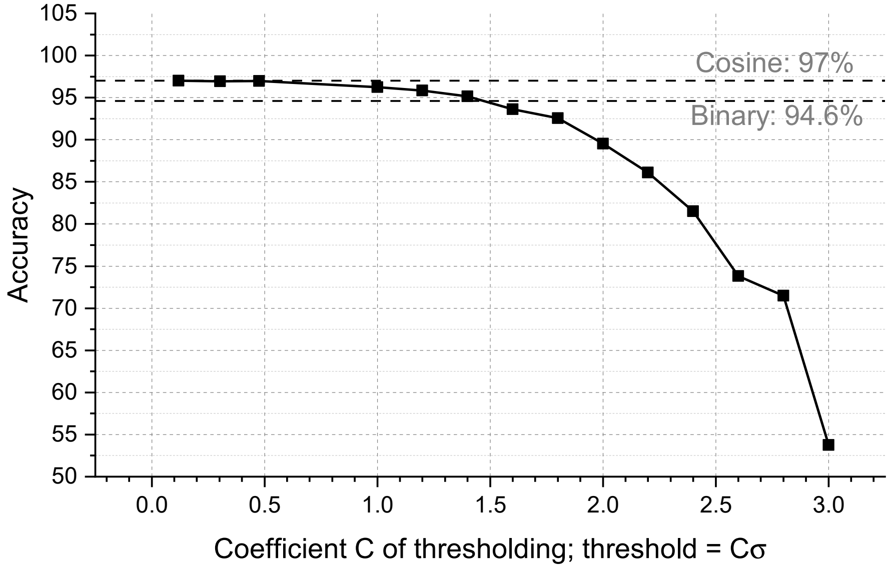
The corollary above provides an expression for the saturation and threshold constants S, T which guarantee that the error in $\mathrm{d_{cos}}$ is at most a chosen ϵ with probability $1 - \delta = 0.99$. The question therefore arises: how does one choose an ϵ which is not specific to the application at hand?

For a generally useful bound, a reasonable choice would be $\epsilon \leq \sqrt{2/\pi}$, the cosine similarity between a normal vector and its bipolarized version in high dimensions [167]. This is because after normalization, one should (at least) be able distinguish the language hyper-vector from its bipolarized version used in the binary model. Using $\epsilon \leq \sqrt{2/\pi}$, lack of confidence $\delta = 0.01$ and HDC dimension $d = 2048$ in Theorem 5.3.1 gives $T \leq \frac{\sigma}{2}\epsilon^2\sqrt{1 - \sqrt{\frac{8}{d}\log 1/\delta}} < 0.3\sigma$.

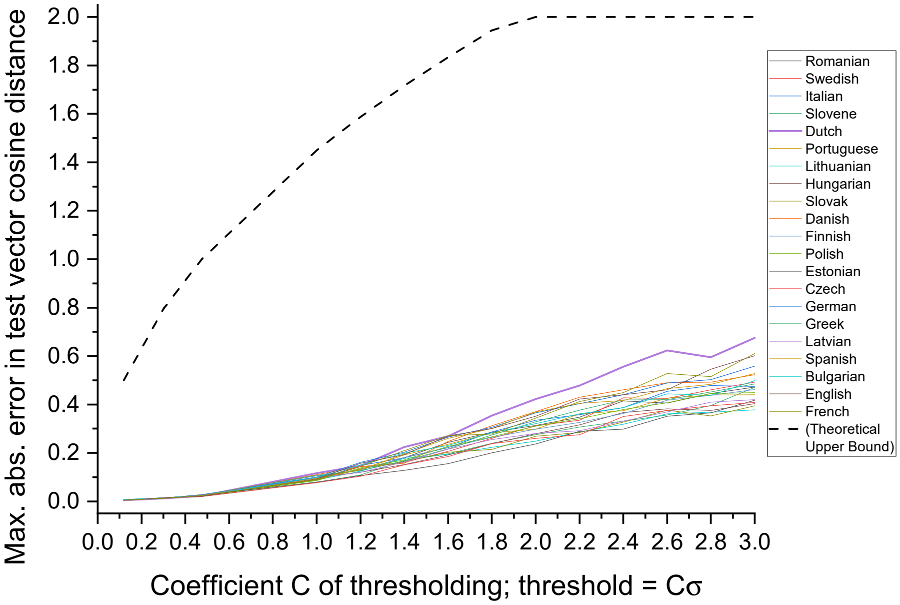
Figure 5-11(b) shows the error in cosine similarity due to thresholding at $T = 0.3\sigma$ so that the error is guaranteed to be at most $\sqrt{2/\pi}$ with probability at least 99%. The Dutch language exhibits the largest absolute error in cosine similarity by thresholding among all languages in the EUROPARL corpus but is consistently $< 0.18\epsilon$. Normalization with $T = 0.3\sigma$ gives 20360 correct of 21000 tests (96.95% accuracy) compared to 20370 correct of 21000 tests for untransformed language hyper-vectors (i.e. 97.00% accuracy).

Putting it all together, saturation and thresholding followed by element-wise integer division by threshold (corollary 5.3.1.1) fixes M to be $\log_2\left\lceil \pi\sqrt{d}\sqrt{\frac{1+\sqrt{(8/d)\log(2/\delta)}}{1-\sqrt{(8/d)\log(2/\delta)}}}\right\rceil + 1$ bits (extra bit is the sign bit in 2's complement representation) and guarantee at most $\epsilon = \sqrt{2/\pi}$ error in cosine similarity with $1 - \delta = 0.99$ probability. For $d = 2048$, this leads to $M = \lceil 7.361 \rceil + 1 = 9$ bits/element. This results in dramatic improvement in bits/element for EUROPARL, from $M = 20$ to $M = 9$ bits and a resultant accuracy drop of at most 0.05%.

Finally, figure 5-12 illustrates that division of inner product by the vector's norm $\|X\|$ is redundant after these transformations: as discussed in section 5.3.1 the threshold $T \propto \sigma \propto \|X\|$. Therefore, the integer division of elements by $T \propto \|X\|$ already achieves this.

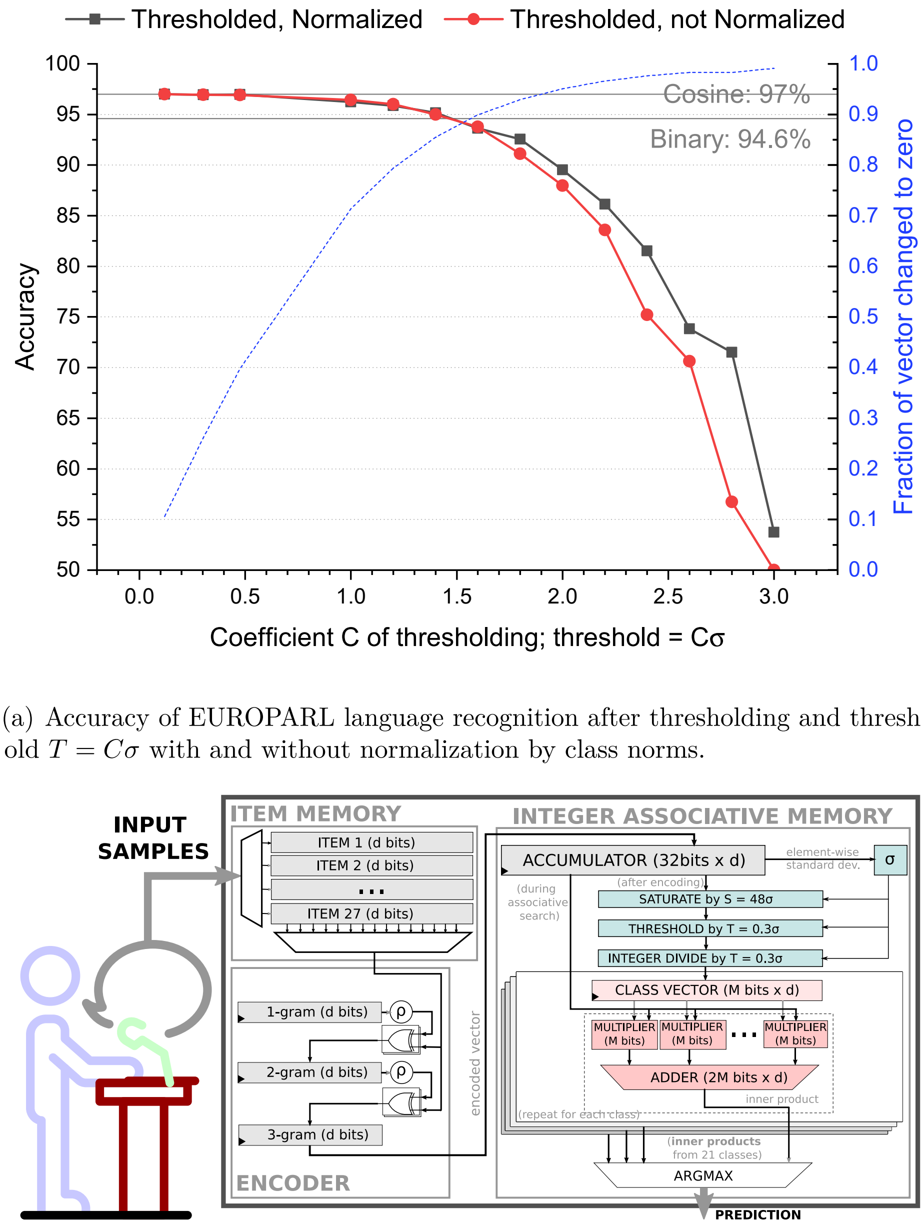


(a) Accuracy of EUROPARL with thresholded language hyper-vectors.



(b) Error in cosine similarity due to thresholding of elements of EUROPARL language hyper-vectors with threshold $T = C\sigma$ is shown. Theoretical upper bound (dotted black line) is derived from corollary 5.3.1.1.

Figure 5-11: Thresholding leads to conservative upper bounds of error introduced in cosine similarity.
(a) As threshold $C\sigma$ increases, EUROPARL language recognition accuracy declines as more elements are zeroed out in the language hyper-vector for EUROPARL.
(b) The maximum error in cosine similarity between all test vectors and each thresholded language hyper-vector for various thresholds T is shown for EUROPARL.



(a) Accuracy of EUROPARL language recognition after thresholding and threshold $T = C\sigma$ with and without normalization by class norms.

(b) Data-path for integer HDC model without division by language vector's norm.

Figure 5-12: Thresholding allows inner product instead of cosine similarity to be compared for associative search.
(a) Using inner product of thresholded language hyper-vectors and test hyper-vector, without division by language vector norm, leads to negligible change in application accuracy. As threshold $C\sigma$ increases, more elements are zeroed out (blue dotted plot line for the right vertical axis) in the language hyper-vector. (b) Unlike the integer HDC data-path without transformations (see figure 5-4(b)), an associative memory supporting saturation, thresholding and element-wise integer division by threshold can directly compare inner product for associative search.

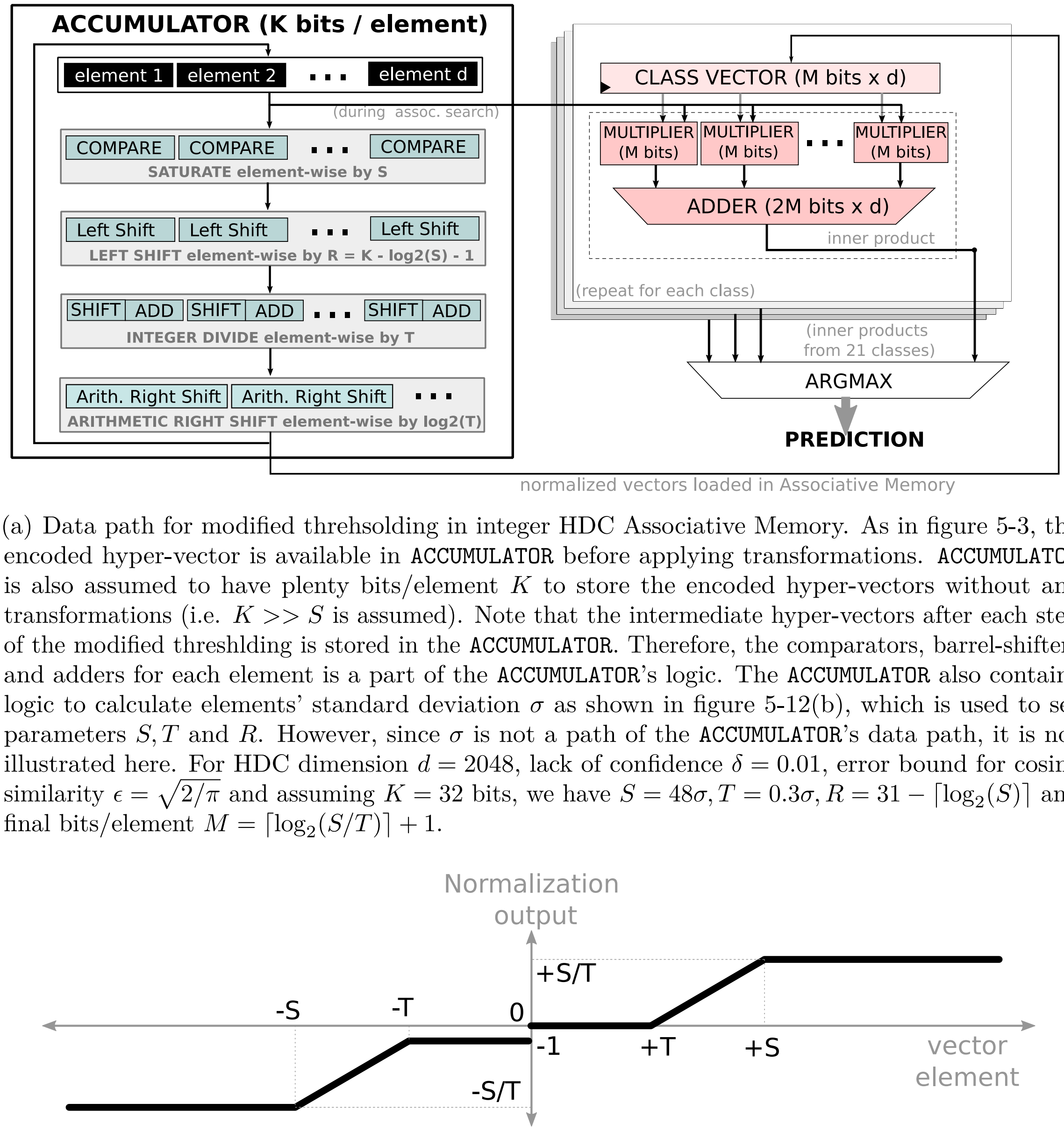
5.3.3 Putting it all together: Modified Thresholding

Figure 5-11(a) shows that if threshold $T = C\sigma$ is too large, the application accuracy using thresholded hyper-vectors collapses. This is because most elements of the hyper-vector become zero. There could be multiple reasons for a large threshold, such as a large and erroneous estimate for standard deviation σ or a small value of ϵ if the application is considered difficult.

A more desirable behaviour for thresholding with large threshold T is that the application accuracy approaches that of the binary HDC model (i.e. using bipolar/binary vectors) instead of dramatically reducing to zero. Like the bipolarization of integer hyper-vectors [87], a really large threshold should *effectively bipolarize* the integer model. Since it cannot be known a priori that the threshold is large, the same thresholding transformation must also guarantee a small loss in cosine similarity as in Theorem 5.3.1. The proposed transformation is called **modified thresholding**.

The key idea behind modified thresholding is to distinguish between positive and negative elements for large thresholds T: small negative elements must become -1 but small positive elements must become 0. Thresholding and distinguishing between small positive and negative elements can be simultaneously accomplished by element-wise integer division of the hyper-vector with threshold T such that the results are rounded *towards* $-\infty$ (i.e. $-1/5 = -1$). While the convention of "*rounding towards negative infinity*" for integer division [168] is adopted by well-known programming languages such as Ruby and Python, rounding towards zero (i.e. $-1/5 = 0$) is more common. However, the rounding towards zero convention does not distinguish between positive and negative numbers of magnitude $< T$. For such dividers, pre-multiplying elements with a large number (such as a power of 2 implementable by left shift), performing the integer division, and *arithmetically right shifting* (i.e. preserving the sign bit) to get the final bits/element $M = \lceil \log_2(S/T) \rceil + 1$ produces the effect of rounding towards $-\infty$. The steps of modifed thresholding and its element-wise transfer function are shown in figure 5-13.

To see that modified thresholding prevents a collapse in application accuracy for a large threshold T, denote the *untransformed* i^{th} language and test hyper-vector as $\mathbb{C}^{(i)}$ and $\mathbb{T}$, and their bipolarized versions $\mathbb{C}^{(i)}_{\pm 1}$ and $\mathbb{T}_{\pm 1}$ respectively. Denote $\mathrm{d_H}(a, b)$ as the hamming distance between bipolar vectors a, b used in the binary model. As $T \uparrow +\infty$, the language hyper-vector $\mathbb{C}^{(i)}$ changes to $(\mathbb{C}^{(i)}_{\pm 1} - 1)/2$ due to modified thresholding by T. Since bipolarization approximately preserves direction of a normal vector in high dimensions [167], we have $\mathrm{d_{cos}}(\mathbb{T}, (\mathbb{C}^{(i)}_{\pm 1} - 1)/2) \approx \mathrm{d_{cos}}(\mathbb{T}_{\pm 1}, (\mathbb{C}^{(i)}_{\pm 1} - 1)/2)$. Next, note that number of 0 and -1 elements are approximately equal as the vector element distribution is approximately centered at 0: the norm $\|(\mathbb{C}^{(i)}_{\pm 1} - 1)/2\| \approx \sqrt{d/2}$ for all classes $\mathbb{C}^{(i)}$. Hence, $\mathrm{d_{cos}}$ with modified thresholding using threshold $T \uparrow \infty$ is identical



(a) Data path for modified threhsolding in integer HDC Associative Memory. As in figure 5-3, the encoded hyper-vector is available in `ACCUMULATOR` before applying transformations. `ACCUMULATOR` is also assumed to have plenty bits/element K to store the encoded hyper-vectors without any transformations (i.e. $K >> S$ is assumed). Note that the intermediate hyper-vectors after each step of the modified threshlding is stored in the `ACCUMULATOR`. Therefore, the comparators, barrel-shifters and adders for each element is a part of the `ACCUMULATOR`'s logic. The `ACCUMULATOR` also contains logic to calculate elements' standard deviation σ as shown in figure 5-12(b), which is used to set parameters S, T and R. However, since σ is not a path of the `ACCUMULATOR`'s data path, it is not illustrated here. For HDC dimension $d = 2048$, lack of confidence $\delta = 0.01$, error bound for cosine similarity $\epsilon = \sqrt{2/\pi}$ and assuming $K = 32$ bits, we have $S = 48\sigma, T = 0.3\sigma, R = 31 - \lceil \log_2(S) \rceil$ and final bits/element $M = \lceil \log_2(S/T) \rceil + 1$.

(b) Transfer function for element-wise modified thresholding. This plot is suggestive – the small steps caused by the discreteness of integers are smoothened for simplicity of depiction.

Figure 5-13: Modified thresholding for the integer HDC Associative Memory.

to the hamming distance based associative search in the binary HDC model:

$$\arg\max_i \mathrm{d}_{\cos}(\mathbb{T}_{\pm1}, (\mathbb{C}_{\pm1}^{(i)} - 1)/2) \approx \arg\max_i \mathbb{T}_{\pm1} \cdot (\mathbb{C}_{\pm1}^{(i)} - 1)/2 = \arg\max_i \mathbb{T}_{\pm1} \cdot \mathbb{C}_{\pm1}^{(i)}$$

$$= \arg\max_i (d - 2\,\mathrm{d}_{\mathrm{H}}(\mathbb{T}_{\pm1}, \mathbb{C}_{\pm1}^{(i)})) = \arg\min_i \mathrm{d}_{\mathrm{H}}(\mathbb{T}_{\pm1}, \mathbb{C}_{\pm1}^{(i)})$$

To summarize, if there aren't enough bits/element when threshold T is too large, the accuracy produced by modified thresholding should approach binary accuracy.

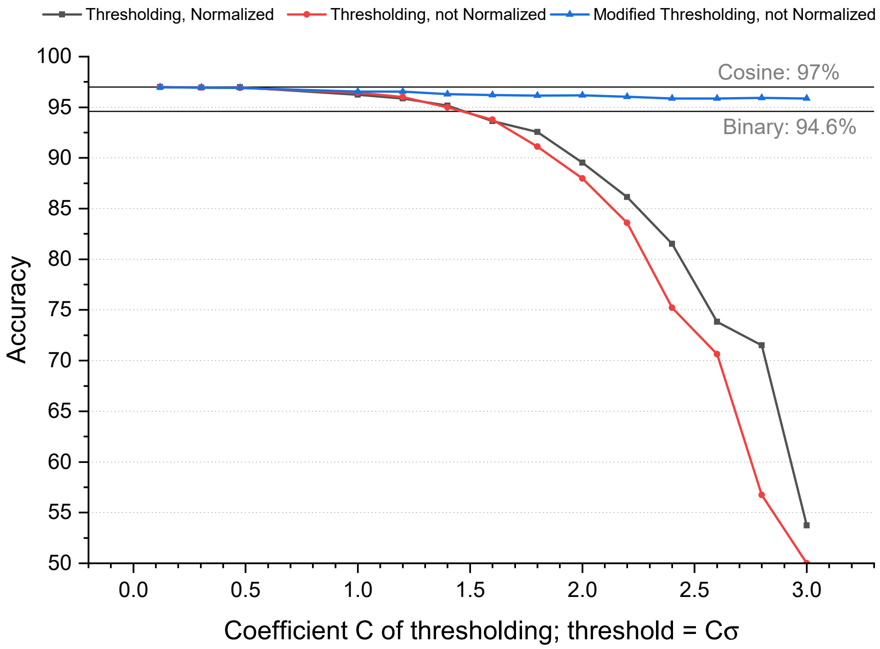


Figure 5-14: Accuracy of EUROPARL language recognition using modified thresholding with increasing threshold $T = C\sigma$.

Figure 5-14(b) shows the 'Modified Thresholding, not Normalized" accuracy plot (symbol: ▲) degrades as threshold T increases, approaching the binary accuracy for language recognition using EUROPARL. However, the degradation is graceful and remains better than the binary accuracy.

For modified thresholding, the error bounds of corollary 5.3.1.1 remains valid. The final bits/element M satisfies $2^{M-1} \approx \pi\sqrt{d}$ for conservatively limiting the error in cosine similarity to a reasonable upper bound of $\epsilon = \sqrt{2/\pi}$. This allows reducing $M = 20$ (97.00% accuracy) to $M = 9$ (96.96% accuracy). The next section examines some estimates for the reduction in hardware complexity of the integer associative memory as a result of the reduced bits/element M.

5.4 Preliminary estimates for hardware savings due to Modified Thresholding

This section estimates the anticipated benefits of using modified thresholding from section 5.3.3 in the integer Associative Memory.

In this preliminary comparison of hardware costs, only the resources required for the *data path* of the integer Associative Memory *during associative search operation* is considered: the flip-flops containing class vectors and class norms (for the untransformed integer HDC model) in the Associative Memory and the query vector in the `ACCUMULATOR`; and logic gates for multipliers, adders and dividers (for the untransformed integer HDC model). A direct comparison of energy cost for associative search is more useful than that for HDC training and applying proposed transformations, as energy per inference is the primary metric of competitiveness for HDC and other learning algorithms [80]. Note that the data path for associative search does not include the transformation operations `SATURATE, THRESHOLD` and `INTEGER DIVIDE`, and the logic for computing and storing element-wise standard deviation σ as shown (in blue) in figure 5-12(b). The compared data-path components are summarized in figure 5-4: for the untransformed Integer Associative Memory (in fig. 5-4 (a)) with $M = 20$ bits/elements for EUROPARL and that after the proposed transformations (in fig. 5-4(b)) with $M = 9$ bits/elements for EUROPARL.

In this study, as shown in figure 5-15 the accounted logic resources are categorized into the number of Flip-Flops (FFs) in bits for sequential logic; number of Half Adders (HAs), number of Full Adders (FAs) and number of Carry Lookahead Blocks (CLBs) for combinational logic. For simplicity, elementary gates of lower complexity such as inverters and AND gates are ignored in this estimate. The total number of transistors in the digital CMOS implementations of the combinational logic gates (HAs, FAs and CLBs) will be the final estimate for overall logic complexity. This simple estimate is likely to correlate well with the actual energy cost of any digital implementation of the data path during associative search. Unless mentioned otherwise, HDC dimension $d = 2048$ is assumed as it is sufficient for most HDC applications [32].

A future study for a detailed accounting of hardware savings could look at the comparison of logic and energy costs of a synthesized or physically-implemented design of the complete HDC processor (both control and data path) for the untransformed Integer HDC model 5-3(b) and that with proposed transformations 5-12(b).

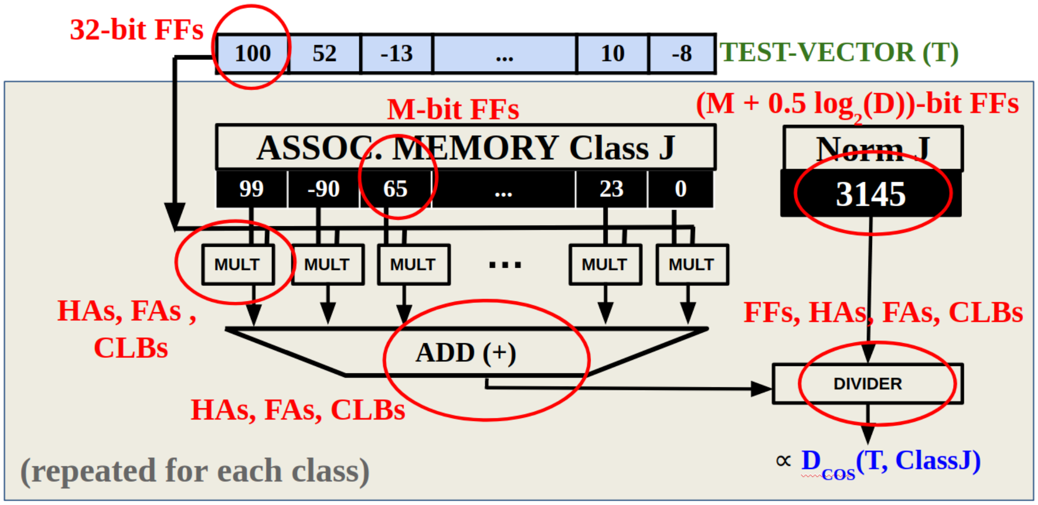


Figure 5-15: Logic components of integer Associative Memory considered for a preliminary estimate of hardware cost.
Only the *data path during the associative search operation* of the integer Associative Memory is considered. The sequential logic is captured as multi-bit Flip-Flops (FFs). The combinational logic is captured as Half-Adders (HAs), Full-Adders (FAs) and Carry Lookahead Blocks (CLBs). The divider logic needed for the *untransformed* Integer Associative Memory requires FFs for storing the dividend (i.e. the inner product of query vector T and stored class vector X), the divisor (i.e. the norm of the class vectors) and the quotient (i.e. the scaled cosine similarity $\overline{\mathrm{d}}_{\cos}(X,T)$ defined in section 5.2). The total number of transistors in digital CMOS implementation of the combinational gates HAs, FAs and CLBs give the final estimate of logic complexity.

5.4.1 Estimating number of sequential gates

It is prudent to account for the number of bits of register storage required in the data path of the integer Associative Memory (in figure 5-15). This sequential storage cost does not change markedly for associative search and transformation operations. The following arithmetic budgets the total number of bits of Flip-Flops (FFs):

1. **ACCUMULATOR.** The `ACCUMULATOR` needs to store the complete, untransformed vector before committing to the Associative Memory. As mentioned in section 5.3, using 32 bits per hyper-vector element for the `ACCUMULATOR` is sufficient for most classification tasks [32], including language recognition on the EUROPARL corpus Therefore, FFs contributed by the `ACCUMULATOR` with and without transformations is $32 \times d = 32 \times 2048 = 65536$ bits.

2. **Associative Memory vectors.** The vectors stored in the Associative Memory consume the majority of storage logic. The EUROPARL language recognition has 21 class hyper-vectors.

   The untransformed integer HDC model requires $M = 20$ bits/element, thus the total number of FF bits are $21 \times M \times d = 21 \times 20 \times 2048 = 860160$ bits.

   After Modified Thresholding in section 5.3.3, $M = 9$ bits/element is sufficient for the transformed language hyper-vectors. Thus, the total number of FF bits are $21 \times 9 \times d = 21 \times 9 \times 2048 = 387072$ bits.

3. **Hyper-vector norms.** The untransformed Associative Memory requires normalization by the norm of the class hyper-vector of the inner product of the query vector and the class hyper-vector (as shown in fig. 5-3(b)). To prevent excessive expenditure of energy during associative search, the class vectors' norms are calculated and stored in registers before the associative search begins. If M bits/element are used for the class hyper-vectors stored in the Associative Memory, $\lceil M + (\log_2 d)/2 \rceil$ bits are required to store the class norm.

   This is only required for the untransformed integer HDC model with $M = 20$; contributing $\lceil 20 + 0.5 \times \log_2 2048 \rceil \times 21 = 26 \times 21 = 546$ bits of FF.

4. **Divider.** Since the query hyper-vector does not have bits/elements larger than that of the class hyper-vectors M, the number of bits required to store the inner-product (i.e. the dividend) is $2M + \log_2 d$ bits. For a slight overestimate, assume another register with the same number of bits to store the quotient of the integer division. Finally, the class norm (i.e. divisor) requires $M + (\log_2 d)/2$ bits of FF. The division is assumed to contain no other sequential elements including pipeline stages.

   Since this is required only for untransformed integer HDC model with $M = 20$, the total number FFs contributed by the divider is $\lceil 5M + (5/2) \log_2 d \rceil$ bits per language hyper-vector $= \lceil 5 \times 20 + 2.5 \times \log_2(2048) \rceil = 128$ bits per language hyper-vector. Since there are 21 languages in EUROPARL, the total bits contributed to FF are $128 \times 21 = 2688$ bits.

To summarize: the untransformed Integer Associative Memory data path for associative search requires $65536 + 860160 + 546 + 2688 = 928930$ bits of FF.

The transformed integer Associative Memory data path for associative search, using Modified Thresholding to get $M = 9$ bits/element, requires $65536 + 387072 = 452608$ bits of FF.

5.4.2 Estimating logic complexity for the adder-tree

In figure 5-15, the **adder tree** is essential to accumulate the element-wise product of query and language hyper-vectors to produce the inner product. Since the language vectors' bits/elements M is larger than the query hyper-vector's bits/element, the adder tree must add d signed integers of $2M$ bits each.

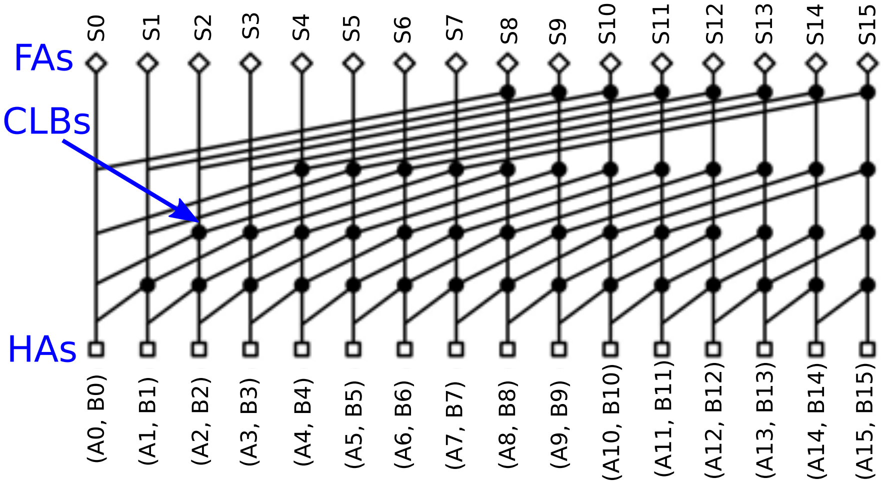


Figure 5-16: The 16-bit Kogge-Stone adder containing Half Adders (HAs), Carry Lookahead Blocks (CLBs) and Full Adders (FAs).
The HAs produce the propagate and generate signals at each bit position, layers of CLBs combine them to produce the sum's propagate and generate signals and the final FAs combine the final propagate and generate signals for sum generation.

A Kogge-Stone adder tree of radix 2 is chosen for the implementation of the adders in this adder tree [169] (illustrated in figure 5-16). The first layer of the Kogge-Stone adder contains Half Adders (HAs) for the production of generate (G) and propagate (P) signals. Subsequent layers contain **Carry Lookahead Blocks (CLBs)** that takes two pairs of generate and propagate signals (G, P) and (G', P') from distinct bit positions and combines them using the logic $(G + P \times G', P \times P')$ where + and × denote logical OR and AND respectively.

Kogge-Stone adder. In the Kogge-Stone adder of radix 2, adding two N-bit numbers to produce a $N + 1$-bit sum, there are $\lceil \log_2(N) \rceil$ layers of CLB reductions The total number of CLBs used across these layers are $\lceil \log_2(N) \rceil N - 2^{\lceil \log_2(N) \rceil} + 1$. The final layer at the output is used to generate the sum bits from propagate P and generate G bits at each bit position produced by the layers of CLB reductions. The final layer has one HA for the LSB and $N-1$ FAs for generating the output bit for the remaining positions.

The tally of combinational gates for Kogge-Stone adder of two N bits numbers are:

- $\lceil \log_2(N) \rceil N - 2^{\lceil \log_2(N) \rceil} + 1$ Carry Lookahead Blocks (CLBs).
- $N+1$ Half Adders (HAs).
- $N-1$ Full Adders (FAs).

Tree of Kogge-Stone adders. Since there are d (a power of 2) N-bit numbers to be added, a binary tree containing $\log_2 d$ stages of addition would minimize the latency of addition. However, the total number of additions is fixed at $d-1$, independent of the strategy chosen for reduction of sums. For addition stage i in the tree, where stage $i=1$ contains $d/2$ adders summing N-bit numbers and stage $i=\log_2(d)$ contains a single adder summing $N+\log_2(d)-1$ bits, $2^{-i}d$ numbers of $N+i-1$ bits are added to produce $2^{-i-1}d$ numbers of $N+i$ bits.

Adder-tree for Integer Associative Memory without transformations. Therefore, the untransformed integer Associative Memory data path for associative search has $N = 2 \times M = 40$ and HDC dimension $d = 2048$. Thus the Kogge-Stone adders required are from $N = 40$ to $N = 51$ (i.e. $\lceil \log_2(N) \rceil = 6$)

- There are $\lceil \log_2 N \rceil = 6$ layers of Carry Lookahead Blocks (CLBs) reduction in each Kogge-Stone adder in the tree. For each stage i of the adder tree, there are $d2^{-i}$ Kogge-Stone adders each contributing $6(N+i-1)-63 = 6i+171$ CLBs. Thus, the total number of CLBs in the Kogge-Stone adder tree for each EUROPARL language are $\sum_{i=1}^{\log_2 d} d2^{-i}(6i+171) = \sum_{i=1}^{11} 2048 \times 2^{-i}(6i+171) = 374535$. Since there are 21 languages in EUROPARL, the total number of CLBs are $21 \times 374535 = 7865235$ CLBs.
- At adder tree stage i, all Kogge-Stone adders contain $N+i = 40+i$ HAs. Thus the total number of HAs in an adder tree for each EUROPARL language are $\sum_{i=1}^{11} 2048 \times 2^{-i}(40+i) = 85963$. Since there are 21 languages in EUROPARL, the total number of HAs are $21 \times 85963 = 1805223$ HAs.
- At adder tree stage i, all Kogge-Stone adders contain $N+i-2 = 38+i$ FAs. Thus the total number of FAs in an adder tree for each EUROPARL language are $\sum_{i=1}^{11} 2048 \times 2^{-i}(38+i) = 81869$. Since there are 21 languages in EUROPARL, the total number of FAs are $21 \times 81869 = 1719249$ FAs.

Since the adder tree is purely combinational and it is presumed there are no additional pipeline stages, this completely summarizes its total logic contribution.

Adder-tree for Integer Associative Memory with Modified Thresholding. Using Modified Thresholding, the integer Associative Memory has transformed vectors with bits/element $M = 9$. Therefore, its data path for associative search has $N = 2 \times M = 18$ and HDC dimension $d = 2048$. Thus the Kogge-Stone adders required are from $N = 18$ to $N = 29$ (i.e. $\lceil \log_2(N) \rceil = 5$)

- There are $\lceil \log_2 N \rceil = 5$ layers of Carry Lookahead Blocks (CLBs) reduction in each Kogge-Stone adder in the tree. For each stage i of the adder tree, there are $d2^{-i}$ Kogge-Stone adders each contributing $5(N+i-1)-31 = 5i+54$ CLBs. Thus, the total number of CLBs in a Kogge-Stone adder tree for each EUROPARL language are $\sum_{i=1}^{\log_2 d} d2^{-i}(5i + 54) = \sum_{i=1}^{11} 2048 \times 2^{-i}(5i + 54) = 130953$ CLBs. Since there are 21 languages in EUROPARL, the total number of CLBs are $21 \times 130953 = 2750013$ CLBs.

- At adder tree stage i, all Kogge-Stone adders contain $N + i = 18 + i$ HAs. Thus the total number of HAs in an adder tree for each EUROPARL language are $\sum_{i=1}^{11} 2048 \times 2^{-i}(18 + i) = 40929$ HAs. Since there are 21 languages in EUROPARL, the total number of HAs are $21 \times 40929 = 859509$ HAs.
- At adder tree stage i, all Kogge-Stone adders contain $N + i - 2 = 16 + i$ FAs. Thus the total number of FAs in an adder tree for each EUROPARL language are $\sum_{i=1}^{11} 2048 \times 2^{-i}(16 + i) = 36835$ FAs. Since there are 21 languages in EUROPARL, the total number of FAs are $21 \times 36835 = 773535$ FAs.

5.4.3 Estimating logic complexity for multipliers

The multipliers of the Associative memory data path (in figure 5-15) are presumed to be implemented as a Wallace tree [152]. Its tree-like structure reduces both the critical path and number of adder gates required [153]. Since the number of bits for the multiplied integers are rather large ($M = 20$ for untransformed Integer HDC model and $M = 9$ for Integer HDC model with Modified Thresholding), the tree-based structure results in substantial hardware savings, especially with respect to a carry-save multiplier [153]. The Wallace-tree based estimate of the multipliers' complexity is likely to correlate strongly with other tree-based implementations. Furthermore, the total number of gates consuming energy in a Wallace-tree multiplier is less than that of any shift-and-add implementation of multipliers.

The simplest implementation of the Wallace tree multiplier of two N-bit integers (as illustrated in [153]) uses Full Adders (FAs) as **3:2 compressors**. An iterative reduction tree containing these compressors converts N^2 partial products into a sum of two $(2N - 2)$-bit numbers. Using Full Adders only instead of a variety of compressors results in about $\log_{3/2} N$ stages of reduction of partial products [153]. Such a reduction step is shown in figure 5-17 for multiplying two 9-bit integers. The first stage computes the partial products $a_i b_j$ i.e. logical AND of bits a_i and b_j for each $i, j \in \{1, 2, \ldots, N\}$.

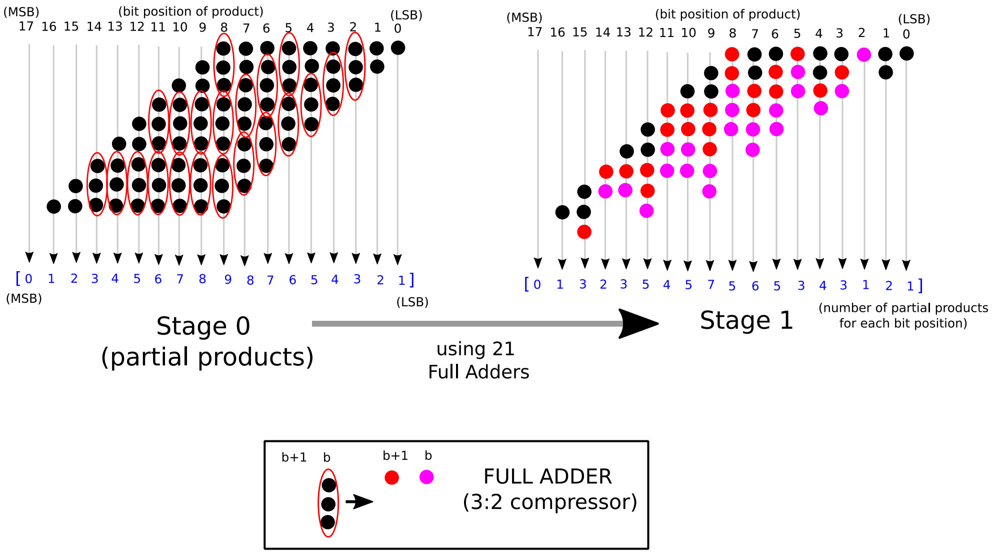


Figure 5-17: A stage of reducing partial products for $N = 9$-bit Wallace tree multiplier using Full Adders.
The first stage produces the partial products $a_i b_j$ which is the logical AND of operand bits a_i and b_j for each $i, j \in \{1, 2, \ldots .N\}$ and uses Full Adders (FAs) to combine specific partial products. For each stage, the number of partial products remaining at every bit position of the product can be summarized in a $2N = 18$-element integer array as shown. The goal of this reduction is make every element of the array ≤ 2.

The array of partial products remaining at each bit position at the end of a partial-product reduction stage will be called **partial product array**. The following sequence of partial product arrays completes the set of reductions for the 9-bit Wallace tree multiplier.

```
Stage 1, 21 FAs: [0,1,3,2,3,5,4,5,7,5,6,5,3,4,3,1,2,1]
Stage 2, 14 FAs: [0,2,1,3,2,4,3,5,4,5,3,4,2,3,1,1,2,1]
Stage 3, 9 FAs: [0,2,2,1,3,3,2,4,3,4,2,2,3,1,1,1,2,1]
Stage 4, 6 FAs: [0,2,2,2,2,1,3,3,2,2,2,3,1,1,1,1,2,1]
Stage 5, 3 FAs: [0,2,2,2,2,2,2,1,2,2,3,1,1,1,1,1,2,1]
Stage 6, 1 FAs: [0,2,2,2,2,2,2,1,2,3,1,1,1,1,1,1,2,1]
Stage 7, 1 FAs: [0,2,2,2,2,2,2,1,3,1,1,1,1,1,1,1,2,1]
Stage 8, 1 FAs: [0,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,2,1]

Total Full Adders = 56
```

Note that the partial product array of the last stage has 0 partial products for MSB of the output, indicating that it is the carry bit from the sum of the two partial products present for the adjacent bit. Similarly, note that the partial product array has only one partial product for the LSB of the output, therefore requiring no addition to generate the output's LSB. All other bits of must be computed by adding together the two numbers formed by the remaining partial products. For this purpose, a Kogge-Stone adder to add two $2 \times 9 - 2 = 16$-bit integers can be used.

Multipliers for Integer Associative Memory with Modified Thresholding. For EUROPARL language recognition, $M = 9$ bits/element is sufficient after transformations. Thus, we can summarize the combinational logic contribution due to multipliers for the Integer Associative Memory with Modified Thresholding as follows:

- 9-bit Wallace-tree multipliers are required, using 56 FAs for each multiplier.
- A 16-bit Kogge-Stone adder may be used for product generation from partial sums, requiring $4 \times 16 - 15 = 49$ CLBs, $16 + 1 = 17$ HAs and $16 - 1 = 15$ FAs (see section 5.4.2).
- Therefore, the total number of logic gates required for each multiplier are: 49 CLBs, 17 HAs and 71 FAs.

Since there are 21 languages in EUROPARL, and given that $d = 2048$ multipliers are required for each language, the total contribution due to multipliers can be determined to be 2107392 CLBs, 731136 HAs and 3053568 FAs.

Similarly, for the untransformed Integer model $M = 20$ bits/element will suffice. This requires 20-bit Wallace-tree multiplier for each hyper-vector element. The sequence of partial product arrays for each reduction stage is shown below:

```
Stage 1, 120 FAs: [0,1,3,2,3,5,4,5,7,6,7,9,8,9,11,10,11,13,12,13,14,
13,11,12,11,9,10,9,7,8,7,5,6,5,3,4,3,1,2,1]
Stage 2, 80 FAs:
[0,2,1,3,2,4,3,5,5,4,6,5,7,6,8,7,9,9,8,9,10,8,9,7,8,6,7,5,5,6,4,5,3,4,2,3,1,1,2,1]
Stage 3, 53 FAs:
[0,2,2,1,3,3,2,4,4,4,3,5,5,4,6,6,6,5,7,6,6,7,5,5,6,4,4,4,5,3,3,4,2,2,3,1,1,1,2,1]
Stage 4, 36 FAs:
[0,2,2,2,2,1,3,3,3,3,2,4,4,4,4,4,3,5,5,4,4,4,4,5,3,3,3,3,4,2,2,2,2,3,1,1,1,1,2,1]
Stage 5, 23 FAs:
[0,2,2,2,2,2,2,2,2,1,3,3,3,3,3,3,2,4,4,3,3,3,3,4,2,2,2,2,2,2,2,2,3,1,1,1,1,1,2,1]
Stage 6, 14 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,3,1,1,1,1,1,1,2,1]
Stage 7, 4 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,2,2,2,3,1,1,1,1,1,1,1,2,1]
Stage 8, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,2,2,3,1,1,1,1,1,1,1,1,2,1]
Stage 9, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,2,3,1,1,1,1,1,1,1,1,1,2,1]
Stage 10, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,3,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 11, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,3,1,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 12, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 13, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 14, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 15, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 16, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 17, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 18, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1]
Stage 19, 1 FAs:
[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1]

Total Full Adders = 342
```

For the $N = 20$-bit Wallace-tree multiplier, a $2N - 2 = 38$-bit Kogge-Stone adder is required for the product generation.

Multipliers for Integer Associative Memory without transformations. Without transformations, $M = 20$ bits/element is sufficient for EUROPARL language recognition. The summary of combinational logic contributed by the multipliers for the untransformed Integer Associative Memory is as follows:

- 20-bit Wallace-tree multipliers are required, using 342 FAs for each multiplier.
- A 38-bit Kogge-Stone adder is needed for product generation from partial sums, requiring $6 \times 38 - 63 = 165$ CLBs, $38 + 1 = 39$ HAs and $38 - 1 = 37$ FAs (see section 5.4.2).
- Therefore, the total number of logic gates required for each multiplier are: 165 CLBs, 39 HAs and 379 FAs.

Since there are 21 languages in EUROPARL, and given that $d = 2048$ multipliers are required for each language, can be determined to be 7096320 CLBs, 1677312 HAs and 16300032 FAs.

5.4.4 Estimating logic complexity for the divider

The divider is required only for the Integer HDC model *without proposed transformations*, requiring at least $M = 20$ bits/element to store the language hyper-vectors of EUROPARL. Recall that since the query hyper-vectors in EUROPARL tests are very unlikely to have bits/elements larger than that of the class hyper-vectors M, the number of bits required to store the inner-product (i.e. the dividend) is $2M + \log_2 d$ bits $= \lceil 40 + \log_2 2048 \rceil = 51$ bits. The vector norm (i.e. the divisor) requires $\lceil M + (\log_2 d)/2 \rceil = 26$ bits. To simplify this at the risk of a slight overestimation, assume that the quotient register is 52 bits wide as well.

Since there is only one integer division per class for the associative search – in contrast to thousands of multiplications and additions – an accurate estimation of an optimized implementation is not necessary for the divider. Therefore, an estimate of logical cost for the shift-and-add divider is sufficient in this context. For dividing a 51-bit dividend with a 26-bit divisor, $51 - 26 + 1 = 26$ subtractions and comparisons need to be performed. The logic required to perform the shifting operation, computing the negative of an integer and comparing against a constant are ignored as they have smaller logical complexity than that of integer addition and subtraction. Furthermore, since the divider's contribution is a tiny minority of the total logic costs of Associative Memory data path, a larger variance in its estimate is permissible. Adding two 51-bit numbers using a Kogge-Stone adder requires $6 \times 51 - 63 = 243$ Carry Lookahead Blocks (CLBs), $51 + 1 = 52$ Half Adders (HAs) and $51 - 1 = 50$ Full Adders (FAs). Therefore, 26 such additions require 6318 CLBs, 1352 HAs and 1300 FAs.

Since EUROPARL has 21 languages where each language hyper-vector requires a division, the total cost due to division is: $21 \times 6318 = 132678$ CLBs, $21 \times 1352 = 28392$ HAs and $21 \times 1300 = 27300$ FAs. The total combinational and sequential costs for the Associative Memory data path during associative search – with and without Modified Thresholding – can now be collected for comparison. This comparison is done in the next sub-section.

5.4.5 Comparison of logic complexity estimates with and without transformations for Integer HDC associative search

This section collects the estimates of the total combinational logic cost (counted as number of CLBs, FAs and HAs) and sequential logic cost (counted as bits of FF) from all components of the Integer Associative Memory's data path during associative search (as shown in figure 5-15) with Modified Thresholding and without transformations. The total number of Complementary MOS transistors in the static digital implementation of Half Adders (HAs), Full Adders (FAs) and Carry Lookahead Blocks (CLBs) are required to estimate the total CMOS transistor count for the combinational logic in Associative Memory's data path.

The Full Adder (FA) contains 28 transistors in a static CMOS implementation (from Fig. 11-4 of [153]). The Half Adder (HA) contains 3 inverters containing 2 transistors each, a XOR gate containing 8 transistors and a NAND gate containing 4 transistors, thereby having a total of 18 transistors. The Carry Lookahead Block can be implemented with 18 transistors as well: 4 inverters with 2 transistors each to invert each of its 4 inputs, a static CMOS gate with 6 transistors to produce the generate output and a static CMOS gate with 4 transistors to produce the propagate output.

Type of Integer HDC Model	FF (# bits)	# HAs	# FAs	# CLBs	# Transistors
(A) Without transformations	928,930	3,510,927	18,046,581	15,094,233	840,197,148
(B) With (modified) Thresholding	452,608	1,590,645	3,827,103	4,857,405	223,223,784
(A)/(B) Improvement ratio	2.05	2.21	4.72	3.11	3.77

Table 5.1: Preliminary estimates for logic cost of associative search in Integer Associative Memory with and without transformations.
HDC dimension is $d = 2048$. The combinational logic of Half-Adders (HAs) containing 18 transistors, Full-Adders (FAs) containing 28 transistors and Carry-Lookahead Blocks (CLBs) containing 18 transistors are combined to produce the total CMOS transistor count shown in the last column.

Table 5.1 summarizes the logic estimates calculated in this section for the Integer Associative Memory's data path. The total CMOS transistor count provides a single metric to compare the amount of combinational logic. These preliminary estimates indicate more than 3.5× savings in logic cost when using Modified Thresholding from section 5.3.3 with only 0.05% drop in accuracy of EUROPARL language recognition.

Chapter 6

A 2048-dim generic Hyper-Dimensional Binary core

A binary HDC processor was manufactured in a 28nm High-K/Metal Gate (HK/MG) process by Taiwan Semiconductor Manufacturing Company (TSMC), Limited. The architectural principles for efficient HDC processor design developed in chapters 3 and 4 were implemented in this chip.

This chapter is divided into three sections. The first section describes the physical characteristics and technical specifications of the fabricated chip. The reader is likely to find these details useful in understanding the complexity of the design process involved – a source of complexity distinct from that regarding the development of its underlying architecture. The second section considers the testing setup and infrastructure. The testing objectives and strategy, test equipments, on-board components and associated software are described. A summary of functionality test results conclude this section.

EUROPARL Language Recognition [19]	19833/21000 correct (94.44%)

EMG hand-gesture recognition of 5 **single degrees-of-freedom gestures** [38]		
Subject 1: 96.93% accuracy	**Subject 2**: 97.93% accuracy	**Subject 3**: 89.87% accuracy

EMG hand-gesture recognition of 20 **single & multiple degrees-of-freedom gestures** [39]									
Experiment 1: For each subject and session, train one trial and test all remaining 4 trials.									**Exp. 2***
Subjects	**Sess. 1**	**Sess. 2**	**Sess. 3**	**Sess. 4**	**Sess.5**	**Sess. 6**	**Sess. 7**	**Sess. 8**	**Sess. 1&2**
Subject 1	94.24%	90.77%	94.99%	99.95%	98.38%	98.29%	95.29%	97.27%	90.08%
Subject 2	90.01%	95.92%	94.35%	97.50%	99.70%	98.89%	99.30%	98.65%	90.22%
Subject 3	89.35%	72.03%	81.32%	97.02%	87.08%	90.18%	87.98%	93.12%	79.83%
Subject 4	87.43%	79.14%	81.53%	77.48%	81.89%	84.26%	86.88%	90.79%	80.45%
Subject 5	72.24%	81.20%	85.28%	85.19%	91.43%	82.30%	74.59%	78.21%	69.59%

Table 6.1: Benchmark applications for measurements on the binary HDC chip. Experiments $1, 2$ and all 8 sessions are described in Supplementary Table 1 of [39]. Experiment 2*: for each subject, use a trial of combined sessions 1 and 2 for training, and test on remaining trials combined for sessions 1 and 2.

The third section contains measurements of the chip for 2 supervised classification tasks listed in table 6.1 from the benchmark of section 4.3.1 in chapter 4. These measurements are compared against other works to establish the chip's energy efficiency in this chapter's conclusion.

As mentioned in section 2.4, the literature on hardware accelerators for Hyper-Dimensional Computing does not contain *any previous works* that have reported a complete HDC processor *entirely on chip.* The results of this chapter establish the energy efficiency and robustness of Hyper-Dimensional Computing reported in numerous simulation studies in the literature (summarized in chapters 1 and 2) using *real-time measurements* for a *complete HDC processor* on chip for the *first time.*

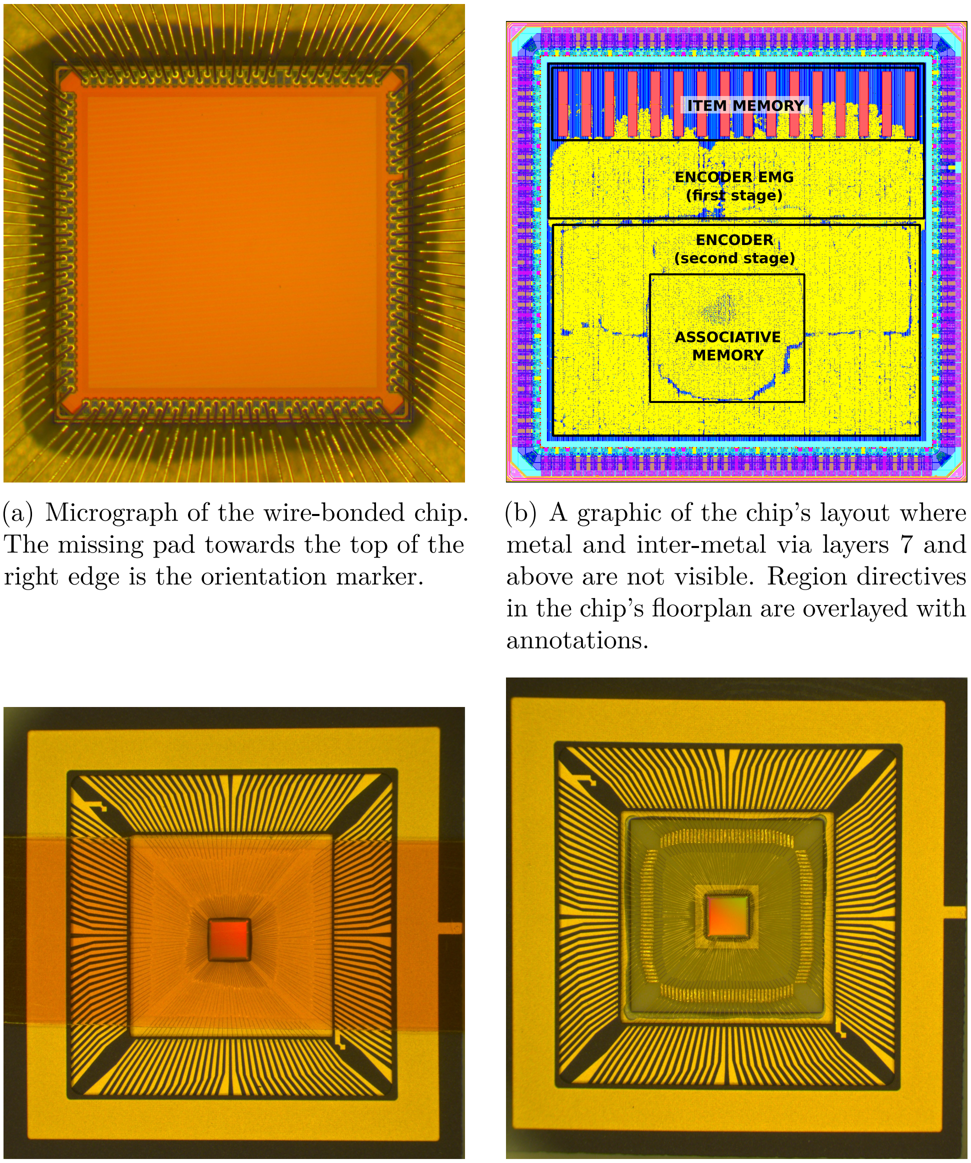
6.1 Physical characteristics and specifications

Figure 6-1 shows the dice bonded to a Ceramic Pin-Grid Array (CPGA) package for testing and measurements. The bonded chip was the final deliverable of all the design, manufacturing and packaging steps that came before.

Design attribute	**Attribute value**
Design area	$2mm$□ die, $1.647mm$□ core, core area utilization is 55%
Technology node	28nm High Performance Mobile (HPM), a High-K/Metal-Gate (HK/MG) process with 10 Copper metal routing layers, 1 poly-silicon routing and 1 Aluminium-Copper redistribution layer.
Manufacturer	Taiwan Semiconductor Manufacturing Company
Nominal supply voltage	0.9V for core logic, 1.8V for Input/Ouput
Setup clock T_{CLK}	$\geq 5.0 \times 10^{-9}$ seconds
Clock jitter assumption	$\leq 10^{-10}$ seconds
On-chip memory	Read-Only Memory (ROM) only. 16 KiB × 16 ROMs = 256 KiB. Used in the Item Memory exclusively.
# cells and transistors	about 4.68 million cells, 23 million MOSFET transistors
# inverters and buffers	$111,068$ inverters and $284,524$ buffers
# flip-flop cells	$216,755$ flip-flop cell instances of different bit-widths

Table 6.2: Summary of technical specifications of the 2048-dim binary HDC processor.

A summary of the chip's design attributes are presented in table 6.2. Containing about *23 million* transistors routed in a core area of about 2.71mm^2, the design size is of medium complexity and its core utilization of 55% – which estimates the difficulty of routing wires – approaches the industry-wide range of $50-60\%$ for mature chip designs. Note that the total number of buffers and inverters, which would be a significant fraction of the total number of cells with unoptimized data-paths or difficult-to-route gate placement, is less than 1% of the total number of cells for this chip. A relatively high clock jitter assumption of 100 picoseconds was considered (as shown in table 6.2) since although its only 2% of the fastest clock period of 5 nanoseconds, the chip does not contain a Phase-Locked Loop (PLL) to synthesize a clock on-chip from an extremely low-jitter, low-frequency crystal clock input. Since the system clock is obtained directly from an input pad, external signal and supply noise will have a greater effect on clock jitter. Consequently, the fastest clock period



(a) Micrograph of the wire-bonded chip. The missing pad towards the top of the right edge is the orientation marker.

(b) A graphic of the chip's layout where metal and inter-metal via layers 7 and above are not visible. Region directives in the chip's floorplan are overlayed with annotations.

(c) The wire-bonded chip in a 144-pin ceramic package. The horizontal tape attaching the lid is also visible.

(d) The wire-bonded chip with silicon interposers in a 144-pin ceramic package.

Figure 6-1: Chip micrograph of the 2048-dim binary HDC processor. The die micrograph (a) and its layout with a few upper layers removed (b) is shown. Two strategies of connecting the die to a 144-pin ceramic pin-grid array were pursued: (c) a classical wire-bond where long gold wires with significantly larger inductive parasitics and (d) an intervening silicon interposer such that bonding gold wires are shorter – resulting in less inductance but possibly greater capacitive parasitics.

of 5 nanoseconds allowed by setup timing constraints could be improved using an on-chip PLL and smaller clock jitter assumptions.

6.1.1 Physical design and implementation

The design was synthesized from a SystemVerilog Register Transfer Logic (RTL) code using Cadence Genus Synthesis 18.10. The synthesized net list was produced using logic gates from the standard cell library provided by TSMC as a part of 28nm HPM technology node's Physical Design Kit (PDK). The synthesized design was placed and routed using Cadence Innovus 18.10. Finally, the placed and routed design was checked for Design Rule Check (DRC) violations so that it can be manufactured by TSMC, Layout Versus Schematic (LVS) checks to confirm that the synthesized net list is logically equivalent to the placed and routed physical geometries for all manufacturing layers, and Electrical Rule Checks (ERC) for verifying the absence of shorts and low-drive strengths in transistors and I/O pads. Calibre DesignReview version 2016.4_15.11 by Mentor Graphics was used to execute the DRC, ERC and LVS scripts. The principal challenges faced in the physical design of the chip were:

- **Good floor planning.** Generating *effective* floor plans is necessary for quick turnaround and high-quality results after place-and-route stage of the synthesized net list subject to design constraints. The main idea is to use region directives for each of the logical modules such that those with high inter-module connectivity are adjacent. Trying to achieve equal density of modules' total cell area to area of of the assigned region also helped in improving floor plans.

  The final floor plan is shown in figure 6-1(b). The Item Memory consists almost entirely of 16 ROMs (visible as 16 vertical bars towards the top edge) and is assigned a smaller region that is contained within that assigned for the first Encoder stage. Similarly, the Associative Memory's region is a part of that for the second Encoder stage. This arrangement also follows from the flow of information from the chip's I/O pad-frame. A simplified explanation is as follows: (most of) the chip's input pads are on the top edge and (most of) the chip's output pads are on the bottom edge. Since data inputs to HDC processors are item addresses to the Item Memory and data outputs are results of the associative search, data can be visualized as *flowing from top to bottom*: input pads from the top edge go into the Item Memory, whose output goes to the two Encoder stages and finally reach the Associative Memory (refer figure 3-5 from chapter 3). The Associative Memory output is transmitted by the output pads on the bottom edge.

  Over an 8 months long design period, executing and evaluating a large number of candidate floor plans was very useful in improving design productivity. It was observed that the design tools consume a large amount of time and memory to produce poor designs when the floor plan is difficult to route, but produce excellent designs using *far smaller runtime memory and time* when the floor plans are effective and well optimized.

- **Viable power plan.** Having a great power plan to produce enough power and ground straps over the entire core area is crucial for a good cell placement by the design tool. Making the power straps too wide or far apart reduces the amount of area available for placement of standard cells. On the contrary, making the power straps too dense uses up valuable routing resources required to route signals and connect logic gates. Therefore, a careful balance must be maintained between access of standard cells to power straps and avoiding overuse of routing tracks. It was also observed that certain standard cells which had input/output ports directly under a *wide* power strap were difficult to route signals to and produced a lot of short violations.

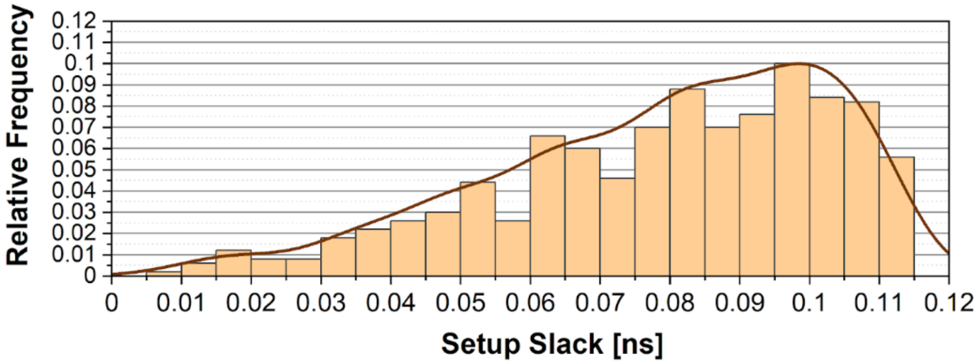


Figure 6-2: Heavy-tailed distribution of positive setup slack.

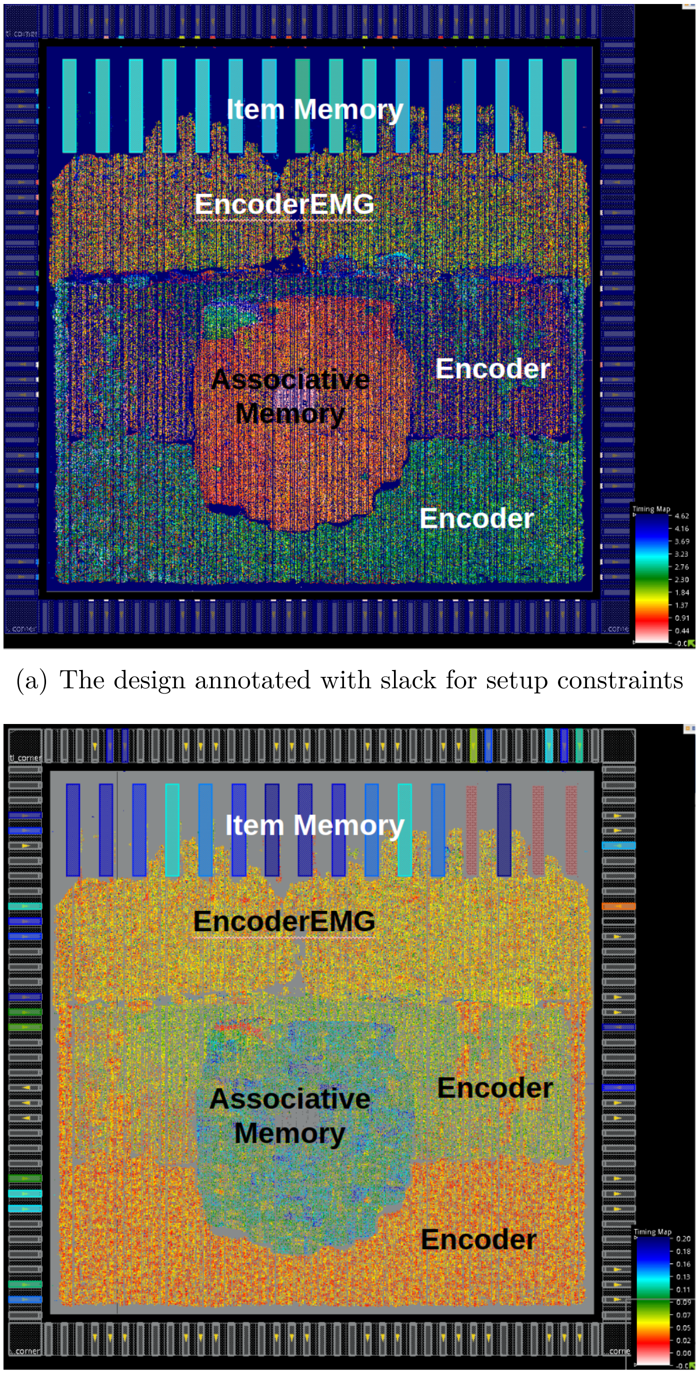
6.1.2 Timing constraints and design convergence

An important objective of this chip is to validate, using *direct on-chip measurements*, the assertion that supply voltage over-scaling leads to greater energy savings but minimal degradation in performance as HDC is inherently robust to representation errors (stated in [18] and demonstrated in figure 1-4). Therefore, a practical consideration in this design is to have fewer data paths failing as VDD is lowered from its nominal value of 0.9 Volts. This is contrary to the objective of timing optimization in the place-and-route EDA algorithms: to achieve the fastest possible clock, the tool searches for a design such that a majority of timing paths have delays *slightly less than the fastest clock period*. Therefore, a highly-optimized design would have a majority of its data paths fail when the core supply voltage VDD is reduced slightly from its nominal value – thereby greatly reducing the overall robustness of the chip's functionality.

To prevent this, a novel timing-optimization strategy was adopted in this chip's design. When compiling the chip's logic into a net list of logic gates from the RTL, a stringent setup clock period constraint of 2.5 nanoseconds was used – this constraint was barely met by Cadence Genus, even after maximum effort configured for its algorithms. Subsequently, when physically implementing the design using Cadence Innovus, the setup clock period constraint was relaxed to $\geq$ 5.0 nanoseconds to encourage negligible timing optimizations by the place-and-route algorithms. The argument is as follows: given a *highly timing-optimized* net list produced during synthesis, a reasonable floor plan with relatively large core utilization (signifying reasonable use of available silicon space) and low total inverter and buffer count (signifying absence of numerous long wires) should produce a good design *without significant timing optimizations.* Furthermore, given that most HDC applications are not timing-critical or ultra-low latency response [32, 80], a slower than optimal setup clock is still useful in a HDC chip.

Indeed, this strategy produced a heavy-tailed distribution of the (positive) setup slack after placement of logic gates and routing of wires was completed. As shown in figure 6-2, the set of paths with a setup slack $\leq +100$ picoseconds have a majority with setup slack close to +100 picoseconds. However, as shown in the colored map in figure 6-3(a) the vast majority of timing paths have a positive setup slack far larger than +100 picoseconds. The great diversity of positive setup slack among *all timing paths* in the design ensures that the chip's functionality will fail gracefully as the core supply VDD is reduced from its nominal value of 0.9 Volts. Note also that the Associative Memory has the largest concentration of near-zero setup slack cells in figure 6-3(a), indicating that *most of the logic failures with lower VDD* is likely to occur in the Associative Memory. Fortunately, the Associative Memory is known to be tolerant to logic errors introduced by VDD over-scaling [18].

Both setup and hold constraints (as shown in figure 6-3(b)) were met for the final design before manufacturing.



(a) The design annotated with slack for setup constraints

(b) The design annotated with slack for hold constraints

Figure 6-3: Standard cells annotated with setup and hold timing constraints. The legend shows the color map and positive slack value in nanoseconds. Both setup and hold constraints were met in the final design.

6.2 Testing infrastructure and experiments

This section describes the peripheral infrastructure for testing the chip's logic, conducting experiments and obtaining measurements of HDC performance and energy efficiency.

6.2.1 Printed Circuit Board and equipments for testing the binary HDC processor

The primary objectives of the Printed Circuit Board (PCB) through which the Binary HDC processor is tested are to house and connect electronic components such that the core and I/O supply voltages (VDD and VDDPST supply rails respectively) can be *precisely controlled and monitored for measuring power consumption.* The PCB is also responsible for connecting the HDC processor chip bonded to the Ceramic Pin-Grid Array (CPGA) package to the appropriate pins of the Field-Programmable Gate Array (FPGA) administering the test harness.

The following features in the PCB circuit are designed to aid in performance and energy efficiency measurements of the HDC processor:

1. **Opal Kelly XEM7310 FPGA.** This is the principal testing infrastructure as the Field-Programmable Gate Array (FPGA) is critical for uploading test programs for measurements, producing a periodic clock signal for the chip's clock input, sending and receiving signals in each clock cycle and reading voltage and current measurements every few clock cycles for the chip's power consumption from VDD and VDDPST voltage supplies. The XEM7310 integration module contains a Xilinx Artix-7 XC7A75T FPGA on-board with necessary power jacks, voltage regulators and a 200 MHz clock crystal [170]. The FPGA produces the chip's clock signal of the desired frequency from the 200 MHz input.

2. **Power Monitoring Integrated Circuits (PMIC).** The `INA229` Power/Energy Monitor by Texas Instruments [171] with Serial Peripheral Interface (SPI) is the primary on-board component for measuring voltage and current drawn for core supply VDD by the HDC processor. Its data sheet mentions a power monitoring accuracy of 0.5%, and energy/charge monitoring accuracy of 1% for ambient temperatures of $-40°C$ to $125°C$. An ultra-precise 0.075 Ohm sense-resistor is used to measure the voltage drop across it and calculate the current drawn from the core supply VDD. The `INA229` chip monitors and updates its internal registers storing bus voltage, shunt voltage across the 0.075 Ohm sense resistor (i.e. VDD rail current) and temperature measurements at a frequency of 6.67 KHz. Calibration using a resistive load (figure 6-4) indicates that sense-resistor's nominal value is close to the nominal value of 0.075 Ohm. As shown in figure 6-5(a), the I/O pins from the FPGA can be used to read these measurement registers using `INA229`'s 5 MHz SPI interface.

3. **Supply voltage switches.** As shown in figure 6-5(a) and (b), two 3-pin male headers determine the source of the HDC processor's VDD (core) and VDDPST

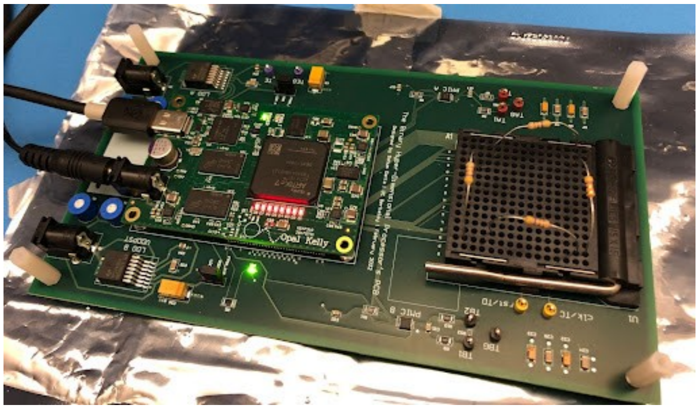


Figure 6-4: Calibrating the sense-resistor for VDD current measurements. Load resistors 5.3, 5.3, 5.3 and 5.2 Ohms were connected in parallel between core voltage supply VDD and common ground GND. The PMIC shunt voltage read 176716 which indicates 55.2 mV. The load resistors draw about 0.758 A from the VDD supply rail, indicating the sense resistor's value is about 0.0728 Ohm.

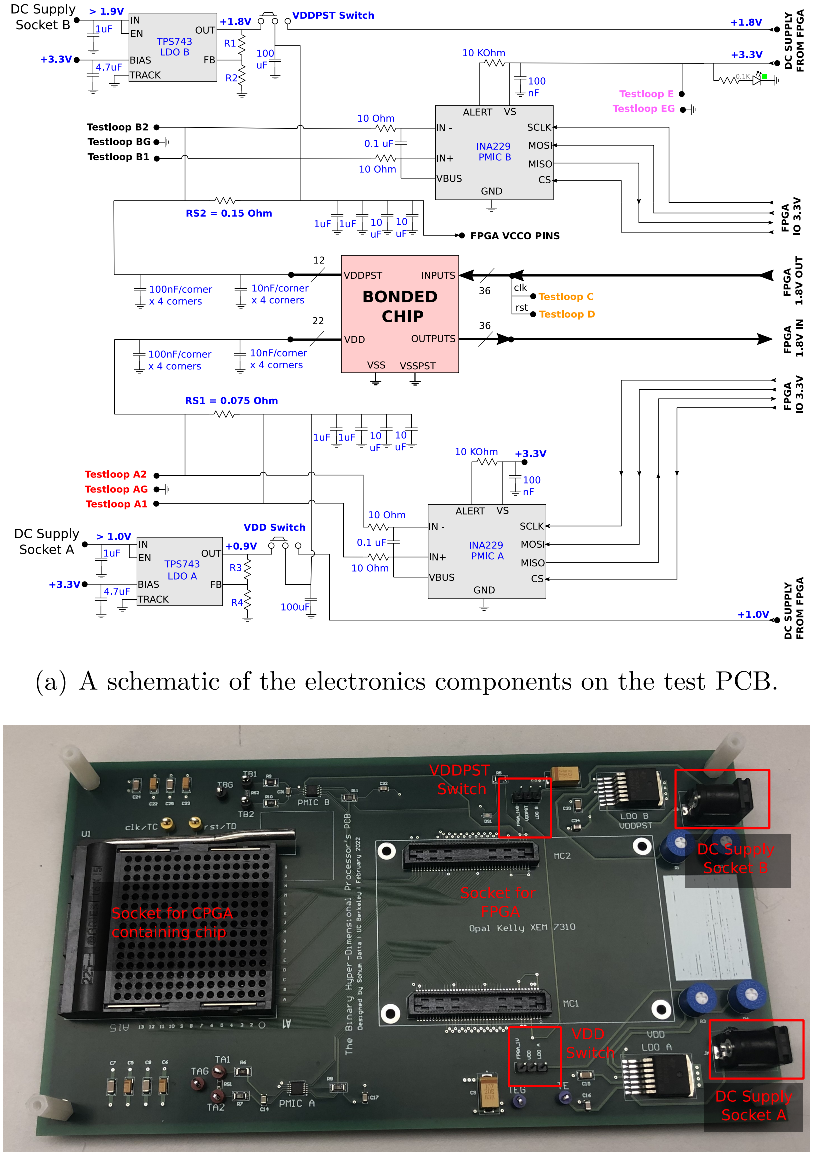
(I/O pad) power supplies. The middle pin corresponds to the selected supply, which can be connected to either the constant regulated output produced by Opal Kelly XEM7310 FPGA's board, or the regulated output of the corresponding LDO from its DC socket input. Finally, the bare middle pin allows a direct power input from a bench-top DC power supply unit, bypassing both FPGA supply and the LDO output.

4. **Low Drop-Out (LDO) voltage regulator.** Texas Instruments' TPS74301 LDOs [172] are used to regulate the voltage input from DC supply sockets in the PCB circuit (see figure 6-5). 3-pin male connectors (labeled in figure 6-5(b)) is used to select voltage inputs from LDO outputs, FPGA's 1.0/1.8 Volt regulated supplies or output of external DC supplies.

5. **Test loops.** Several test loops are provided on the PCB (see figure 6-5(b)). They help in observing and troubleshooting signals *in real time* as the test program is being applied on the chip.

Figure 6-6 shows the general setup for running tests and measurements. A personal computer acts as a host to the Opal Kelly FPGA, uploading the test program and instructing the begin and end of the test sequence. Opal Kelly's FrontPanel Software Development Kit (SDK) [173] was used to command the execution of FPGA's test logic using a Graphical User Interface (GUI) in the laptop host. As shown in figure 6-6(b), the USB cable connecting FPGA to laptop host is the main link of communication between the PC host and the FPGA controlling the test program.

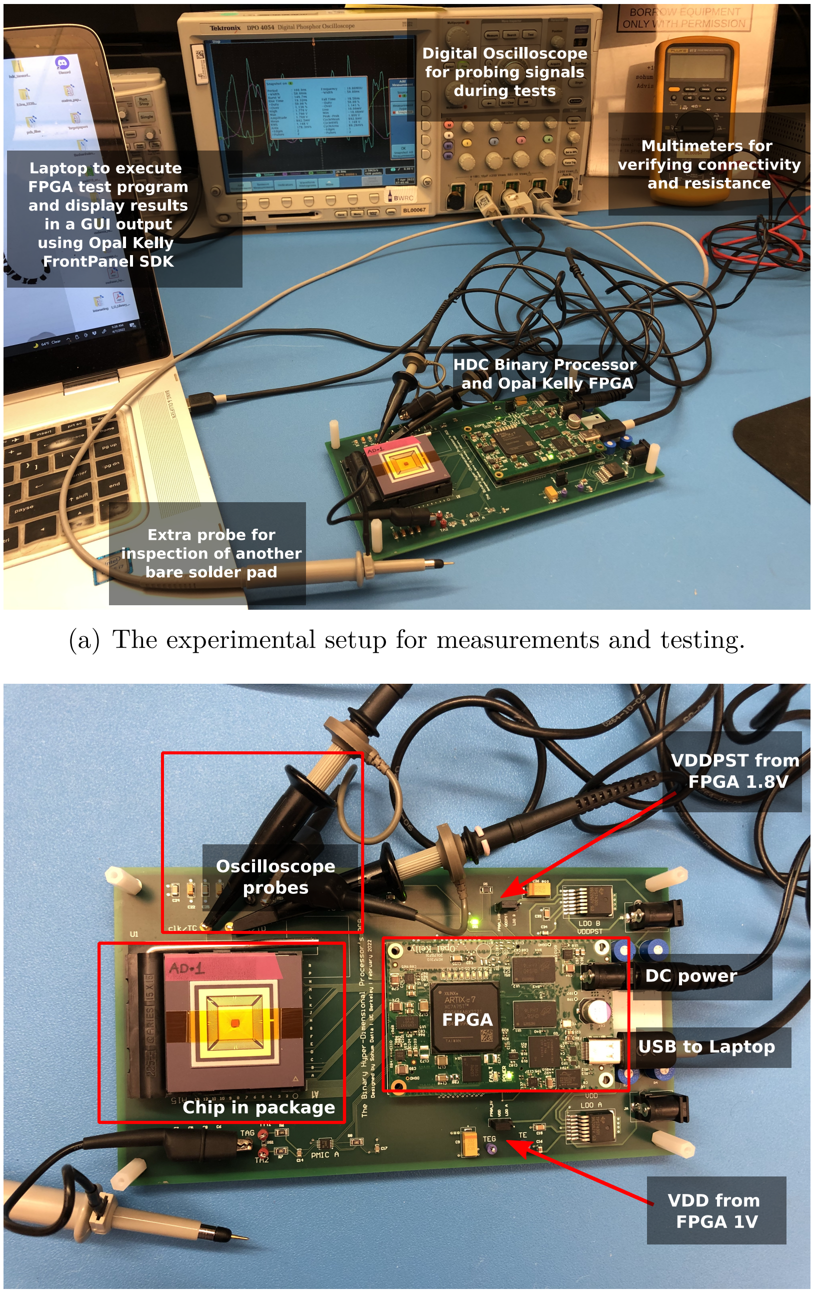
The next sub-section describes a few functionality tests establishing the proper functioning of the chip.



(a) A schematic of the electronics components on the test PCB.

(b) The test PCB with its electronic components.

Figure 6-5: The test Printed Circuit Board and components on board.
(a) Two Low-Drop-Out voltage regulators (LDOs) and Power Monitoring ICs (PMICs) for the supply VDD (for core) and VDDPST (for I/O pads) are present in PCB's testing circuit. Apart from the FPGA and the HDC processor, these are the only other active components. (b) Components on the soldered PCB is shown. The VDDPST and VDD switches as 3-pin male header posts allow the user to provide a regulated supply from a bench-top DC Power supply as well.


BORROW EQUIPMENT ONLY WITH PERMISSION
Digital Oscilloscope for probing signals during tests
Multimeters for verifying connectivity and resistance
Laptop to execute FPGA test program and display results in a GUI output using Opal Kelly FrontPanel SDK
HDC Binary Processor and Opal Kelly FPGA
Extra probe for inspection of another bare solder pad
VDDPST from FPGA 1.8V
Oscilloscope probes
DC power
FPGA
USB to Laptop
Chip in package
VDD from FPGA 1V

(a) The experimental setup for measurements and testing.

(b) PCB test setup.

Figure 6-6: The test setup and equipments for experiments and measurements. Oscilloscopes, PC host and a multi-meter are necessary for performing tests.

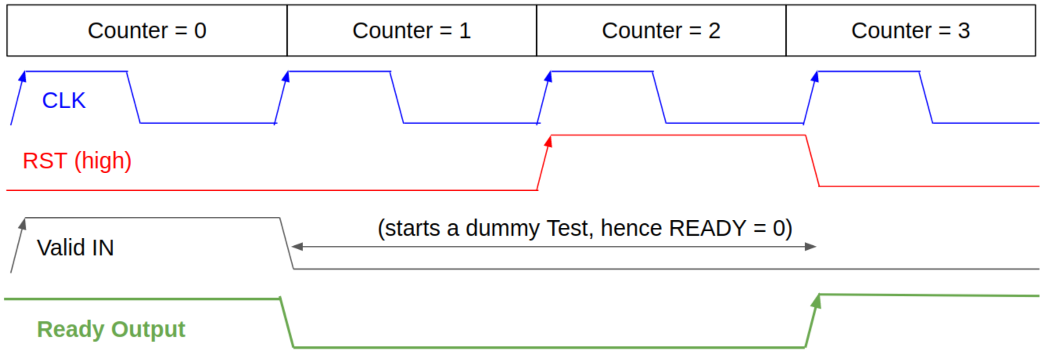
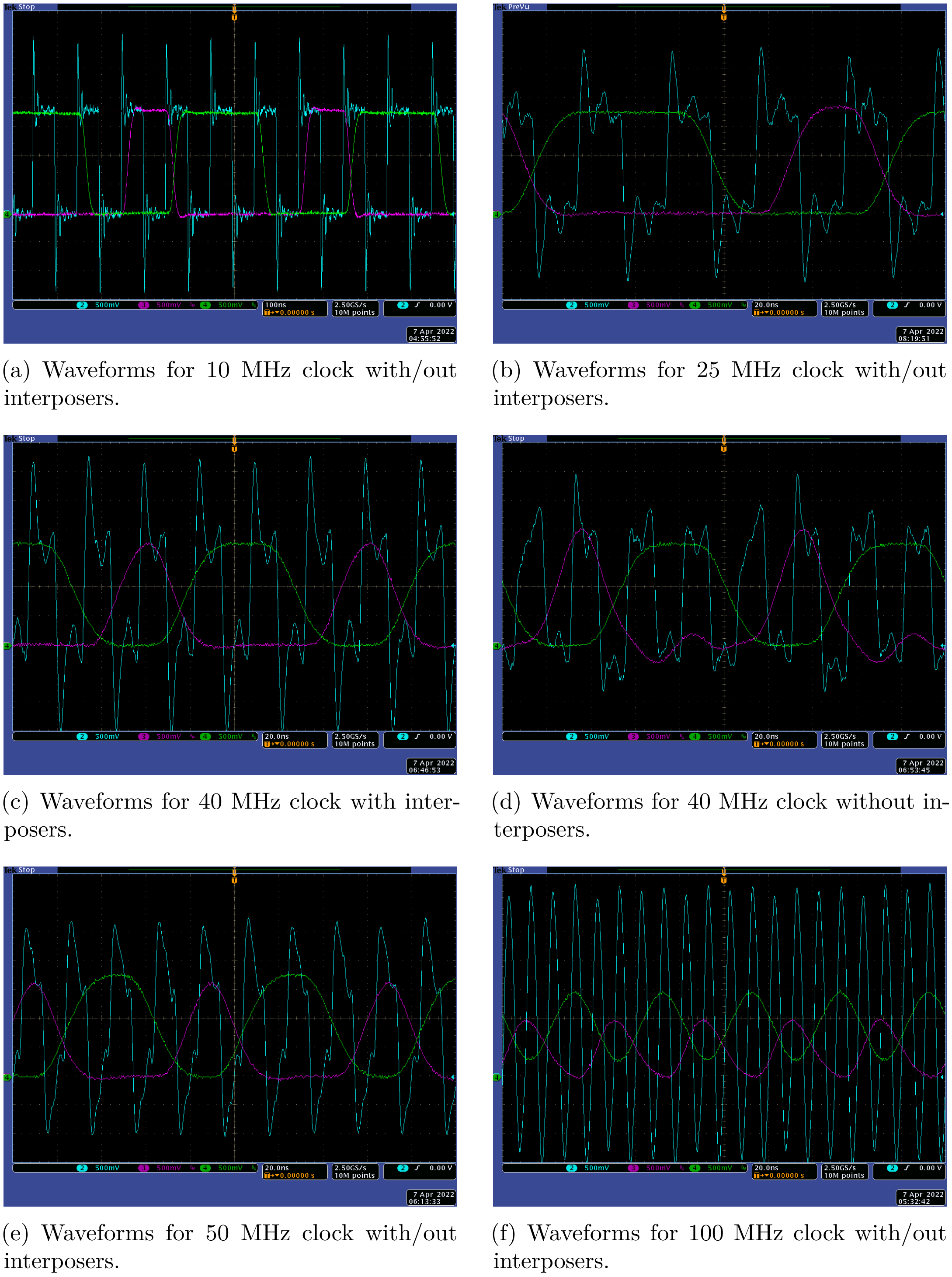


Figure 6-7: Waveforms for the correct behavior when testing the ready output. The 2-bit counter repeatedly counts up from 0 to 3, periodically toggling the `valid_in` and `rst` inputs to effect periodic changes in output `ready` as shown.

6.2.2 Testing basic I/O and chip response

Tests verifying the functionality of the HDC processor chip are described. The first test verifies correct behavior of the `ready` output when the synchronous reset-high input `rst` and *least significant bit* of valid input `valid_in` is toggled. This `ready-valid_in` request-response test establishes proper receipt of input signals from the FPGA to the chip, correct functioning of the chip's primary control logic and proper delivery of chip's output signal to the FPGA.

The waveforms of inputs `clk`, `rst`, `valid_in` and output `ready` are captured by a digital oscilloscope. Waveforms are an illustration of the correct functioning of the chip's `ready` output. The default `ready` output by the chip after reset is HIGH, indicating that it is ready for the next computation to commence. As shown in figure 6-7, when the `valid_in` pad is asserted HIGH while reset `rst` is LOW, the next clock cycle onwards should see the output `ready` de-asserted by the chip as a valid computation should have commenced. Consequently, when output `ready` is de-asserted by the chip, the reset `rst` is asserted HIGH indicating a synchronous global reset in the next cycle. Thus, the next cycle should see the output `ready` asserted again as the ongoing valid computation should have been terminated by a global reset. Figure 6-8 lists the output waveforms and **confirms correct functionality of the HDC processor's basic I/O**. For all except 40 MHz clock, no differences were observed in the waveforms for chips bonded to package with or without inter-posers (refer figure 6-1(c), (d)). For 40 MHz clock, the waveforms of chips bonded classically with long bond wires showed higher distortions due to greater inductive parasitics. Since the rise and fall times of the `ready` output was always $\approx$ 19 nanoseconds, only frequencies $\leq$ 25 MHz are admissible as clock is slow enough to ensure that output transitions complete while the oscilloscope's 8pF probes are attached to the test-loops.



(a) Waveforms for 10 MHz clock with/out interposers.

(b) Waveforms for 25 MHz clock with/out interposers.

(c) Waveforms for 40 MHz clock with interposers.

(d) Waveforms for 40 MHz clock without interposers.

(e) Waveforms for 50 MHz clock with/out interposers.

(f) Waveforms for 100 MHz clock with/out interposers.

Figure 6-8: Captured waveforms for testing ready output.
The waveforms for chip inputs clock (clk), reset (rst) and output ready are shown. A digital phosphor oscilloscope was used with a 8pF, 500 MHz voltage probe.

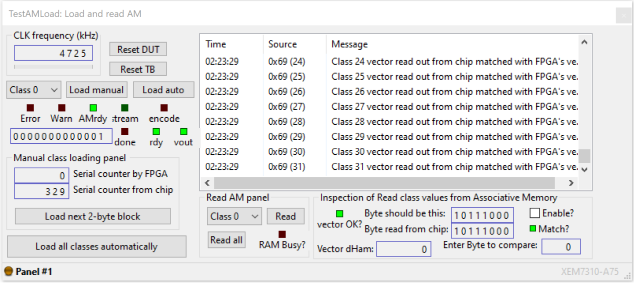
6.2.3 Testing Associative Memory functionality.

Testing functionality of the Associative Memory is critical to test the functional logic of the chip's output pads. Given that data output pads are connected directly to the Associative Memory, a faulty Associative Memory control logic or configuration state could interfere with HDC processor's operations which may (not) have any computation performed in the Associative Memory. Therefore, a complete functional verification of the Associative Memory is of the highest priority among the 3 major components of the generic HDC architecture (as shown in section 3.3.3).

The Associative Memory functionality tests are split into the following tasks *performed sequentially*:

1. **Check that hyper-vectors are loaded successfully.** Using the `LOAD_AM` mode, a randomly-generated hyper-vectors may be inputted into the Associative Memory and loaded to any address $0-31$ by inputting 2 bytes/cycle in little-endian order. After the loading is complete, the chip should assert the `done` output for a *single clock cycle* to signal a successful vector load operation.
2. **Check that loaded class hyper-vectors are read out successfully.** The `READ_AM` mode is used to read out the contents of the Associative Memory from any address $0-31$. Once asserted, the chip commences a data transfer of 1 byte/cycle in little-endian order including that address's `class valid` bit. Once all the 256 bytes are outputted by the chip, the `done` output should be asserted for a *single clock cycle* to signal a successful vector read operation. All outputted vector should match the corresponding loaded vectors *exactly*.
3. **Check that an associative search is working correctly.**
   The mode `LOAD_AND_COMPARE_AM` is used to load a vector 2 bytes/cycle (in little-endian order) and perform associative search for among all vectors loaded. The HDC processor *is supposed to* automatically change its internal control state to stop loading two-byte blocks once 128 valid blocks have been received to commence the associative search operation.
   When the associative search is completed, the `done` output should be asserted by the chip in the next clock cycle. The address of the closest match (the largest address in case of a tie) is outputted by the bus `label_out`, and the hamming distance of the query hyper-vector with the closest match is outputted at output bus `best_distance`. Both outputs are held fixed until the next valid computation is asserted.
   For all hyper-vectors stored in the Associative Memory, the output hamming distance should be 0 and the output `label_out` should be the hyper-vector's own address. Agreement indicates Associative Memory's distance and minimum-argument logic is working correctly for *each closest-match address location.*

Using Opal Kelly's FrontPanel SDK [173], it is easier to produce a GUI that can begin these tasks, collect the results and display messages summarizing the outcome of the test. This is shown in figure 6-9 (a), where the GUI for the testing program for loading and reading out random hyper-vectors into Associative Memory was developed. All 32 random hyper-vectors were *successfully loaded, read out and matched.* Furthermore, figure 6-9 (b) confirms that the loaded hyper-vectors, when



TestAMLoad: Load and read AM

CLK frequency (kHz): 4725

Class 0

```
Time      Source     Message
02:23:29  0x69 (24)  Class 24 vector read out from chip matched with FPGA's ve.
02:23:29  0x69 (25)  Class 25 vector read out from chip matched with FPGA's ve.
02:23:29  0x69 (26)  Class 26 vector read out from chip matched with FPGA's ve.
02:23:29  0x69 (27)  Class 27 vector read out from chip matched with FPGA's ve.
02:23:29  0x69 (28)  Class 28 vector read out from chip matched with FPGA's ve.
02:23:29  0x69 (29)  Class 29 vector read out from chip matched with FPGA's ve.
02:23:29  0x69 (30)  Class 30 vector read out from chip matched with FPGA's ve.
02:23:29  0x69 (31)  Class 31 vector read out from chip matched with FPGA's ve.
```

0000000000001

Manual class loading panel

0 Serial counter by FPGA

329 Serial counter from chip

Read AM panel: Class 0

Inspection of Read class values from Associative Memory

Byte should be this: 10111000

Byte read from chip: 10111000

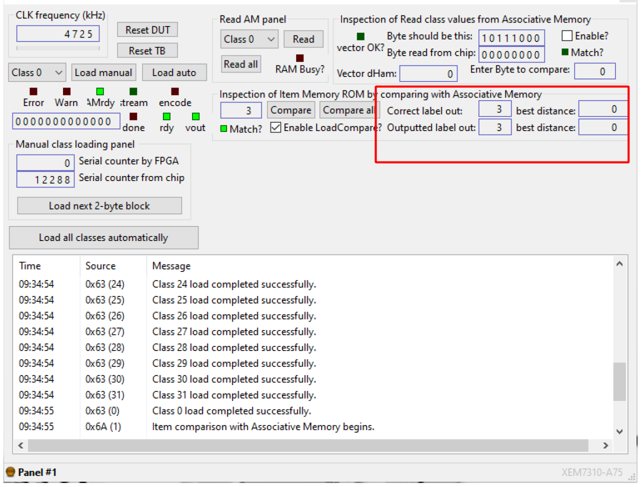
☐ Enable?

🟩 Match?

Vector dHam: 0

Enter Byte to compare: 0

(a) GUI to load, read out and compare vectors into Associative Memory.



CLK frequency (kHz): 4725

Class 0

0000000000000

Manual class loading panel

0 Serial counter by FPGA

12288 Serial counter from chip

Read AM panel: Class 0

Inspection of Read class values from Associative Memory

Byte should be this: 10111000

Byte read from chip: 00000000

☐ Enable?

🟩 Match?

Vector dHam: 0

Enter Byte to compare: 0

Inspection of Item Memory ROM by comparing with Associative Memory

3

🟩 Match?

☑ Enable LoadCompare?

Correct label out: 3 best distance: 0

Outputted label out: 3 best distance: 0

```
Time      Source     Message
09:34:54  0x63 (24)  Class 24 load completed successfully.
09:34:54  0x63 (25)  Class 25 load completed successfully.
09:34:54  0x63 (26)  Class 26 load completed successfully.
09:34:54  0x63 (27)  Class 27 load completed successfully.
09:34:54  0x63 (28)  Class 28 load completed successfully.
09:34:54  0x63 (29)  Class 29 load completed successfully.
09:34:54  0x63 (30)  Class 30 load completed successfully.
09:34:54  0x63 (31)  Class 31 load completed successfully.
09:34:55  0x63 (0)   Class 0 load completed successfully.
09:34:55  0x6A (1)   Item comparison with Associative Memory begins.
```

(b) GUI to load and associatively search vectors into chip's Associative Memory.

Figure 6-9: The FrontPanel testing GUI to load, read-out and associatively compare hyper-vectors into the HDC processor's Associative Memory.
(a) Messages indicate all 32 random hyper-vectors were loaded successfully, read out successfully and all 32 vector matched. (b) The red box and asserted "Match?" LED confirms that the chip's associative search outputs are correct.

associatively searched among all stored hyper-vectors in Associative Memory, *always returns* itself as the closest match with a hamming distance of 0 and its own address as output.

Therefore, the Associative Memory is functioning correctly.

6.2.4 Testing ROM and Item Memory functionality.

Since the Read-Only Memory (ROM) in the HDC processor's Item Memory is the *only source* of item hyper-vectors in the chip, it is crucial to check the funcionality of the ROM and Continous Item Memory (as described in section 4.2.1).

The `READ_IM` mode is used to read out hyper-vectors produced by Item Memory at the rate of 1 byte/cycle in little-endian order. When the read-out is complete, the HDC processor should assert the `done` output signal for a single clock cycle.

1. **Check the ROM hyper-vectors.** The first part of the Item Memory tests is to download and inspect the ROM item vectors. The item vectors were downloaded and verified [1] against multiple die for consistency across chips. These items were used to produce the accuracy numbers for EUROPARL language recognition and EMG hand-gesture recognition benchmarks listed in table 6.1.
2. **Check the Continuous Item Memory**. For each of the 1024 ROM items as the origin vector (`V0` in section 4.2.1), all 1024 continuous items were generated and its hamming distance to the origin vector verified. This step confirms that all possible item hyper-vectors are generated correctly by the chip's Item Memory, as shown in figure 6-10(a)
3. **Check associative search outputs with each ROM item.** In this final test, each of the 1024 ROM items are associatively searched among 32 random hyper-vectors loaded in the Associative Memory. The Encoder is configured to its post-reset default: The first stage is disabled and the second stage is enabled and programmed to be a delay chain. As shown in figure 6-10(b), all comparisons produced outputs that matched the expected values.

All Item Memory tests were passed *successfully*.

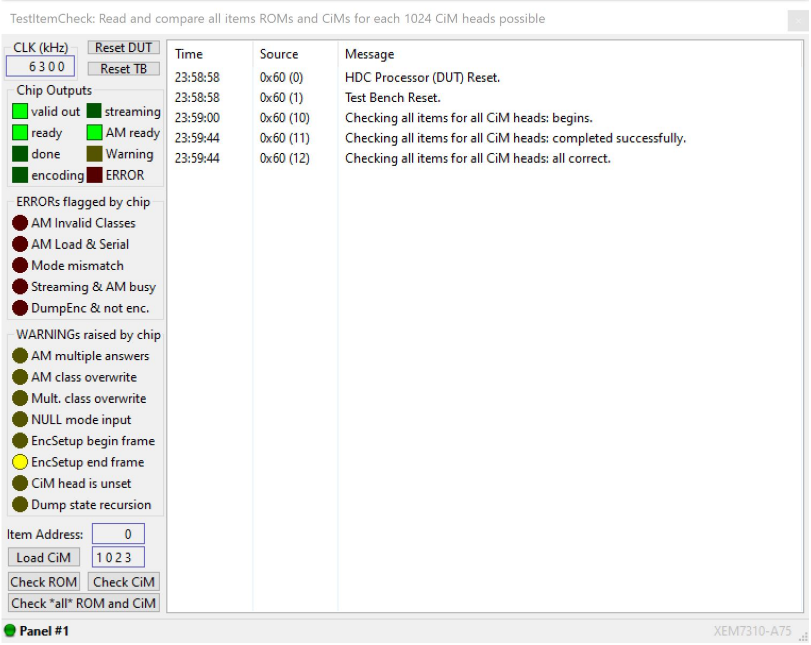
It has been verified that the chip's Item Memory correctly produces items for each item address, which travel *unaltered* through the default Encoder configuration (i.e. a delay chain) and results in correct Associative Memory search behavior.

6.2.5 Testing Encoder for on-chip benchmark applications

Before obtaining energy efficiency measurements, the only remaining functionality test is to check whether the Encoder can successfully encode vectors for EMG hand-gesture recognition (a 2-stage 5-gram encoding) and EUROPARL hand-gesture recognition (a 3-gram encoding) as listed in table 6.1. From the Item Memory functionality tests, the delay chain for Encoder's second stage has already been verified.

While the functionality of the Encoder can be tested more elaborately, directly verifying with the application-based encodings is a composite method of verifying functionality. It also saves time as a correct functioning obviates the need for detailed

[1] All 1024 item hyper-vectors downloaded from the chip's ROM were *not equal* to those originally designed to be programmed into the ROM. In fact, a careful analysis revealed that these item vectors were (statistically) random and independent from the intended ROM hyper-vectors. Owing to the fact that any set of randomly-generated hyper-vectors could serve as items, these new vectors were adopted as the de-factor Item Memory of the fabricated processor. These vectors may have been generated inadvertently while finalizing the design data-base for manufacturing.



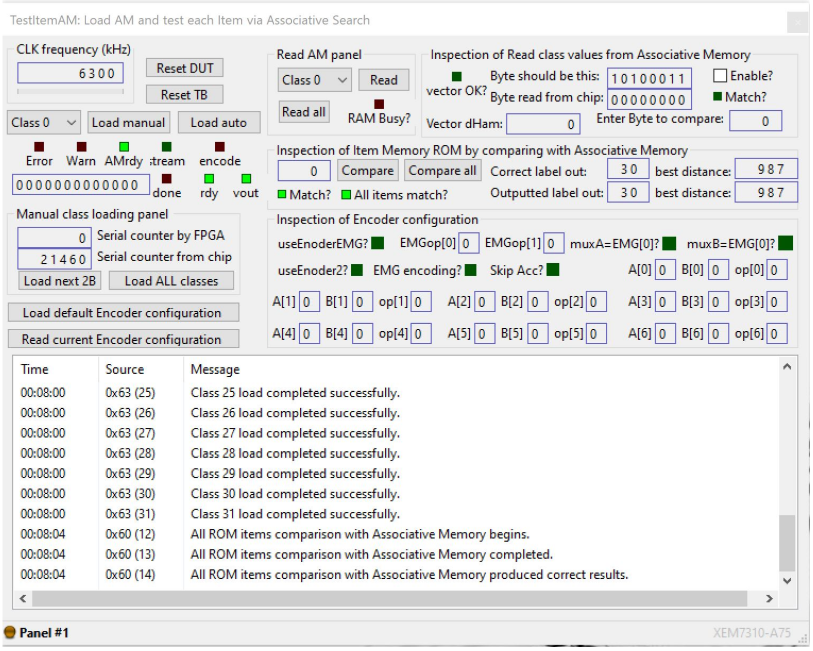
CLK (kHz): 6300

Item Address: 0

Load CiM: 1023

```
Time      Source     Message
23:58:58  0x60 (0)   HDC Processor (DUT) Reset.
23:58:58  0x60 (1)   Test Bench Reset.
23:59:00  0x60 (10)  Checking all items for all CiM heads: begins.
23:59:44  0x60 (11)  Checking all items for all CiM heads: completed successfully.
23:59:44  0x60 (12)  Checking all items for all CiM heads: all correct.
```

(a) GUI to check all ROM and continous items.


TestItemAM: Load AM and test each Item via Associative Search
Reset DUT
Reset TB
Load manual
Load auto
Error
Warn
AMrdy
:tream
encode
done
rdy
vout
Manual class loading panel
Load next 2B
Load ALL classes
Load default Encoder configuration
Read current Encoder configuration
Read AM panel
Read
Read all
RAM Busy?
Inspection of Read class values from Associative Memory
vector OK?
Enable?
Match?
Inspection of Item Memory ROM by comparing with Associative Memory
Compare
Compare all
Match?
All items match?
Inspection of Encoder configuration
useEnoderEMG?
muxA=EMG[0]?
muxB=EMG[0]?
useEnoder2?
EMG encoding?
Skip Acc?
Panel #1
XEM7310-A75

CLK frequency (kHz): 6300

Class 0

000000000000

0 Serial counter by FPGA

21460 Serial counter from chip

Class 0

Byte should be this: 10100011

Byte read from chip: 00000000

Vector dHam: 0

Enter Byte to compare: 0

0

Correct label out: 30 best distance: 987

Outputted label out: 30 best distance: 987

EMGop[0] 0 EMGop[1] 0

A[0] 0 B[0] 0 op[0] 0

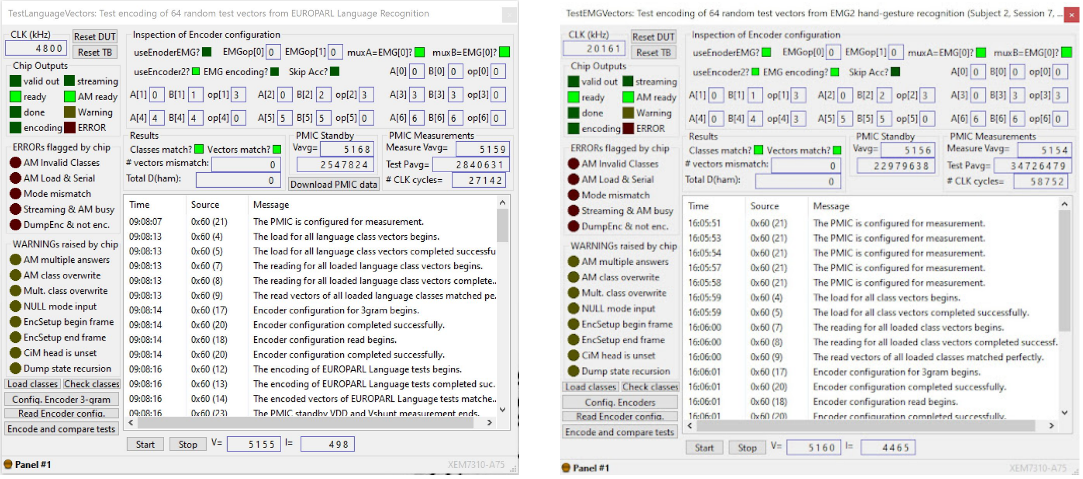
A[1] 0 B[1] 0 op[1] 0 A[2] 0 B[2] 0 op[2] 0 A[3] 0 B[3] 0 op[3] 0

A[4] 0 B[4] 0 op[4] 0 A[5] 0 B[5] 0 op[5] 0 A[6] 0 B[6] 0 op[6] 0

```
Time      Source     Message
00:08:00  0x63 (25)  Class 25 load completed successfully.
00:08:00  0x63 (26)  Class 26 load completed successfully.
00:08:00  0x63 (27)  Class 27 load completed successfully.
00:08:00  0x63 (28)  Class 28 load completed successfully.
00:08:00  0x63 (29)  Class 29 load completed successfully.
00:08:00  0x63 (30)  Class 30 load completed successfully.
00:08:00  0x63 (31)  Class 31 load completed successfully.
00:08:04  0x60 (12)  All ROM items comparison with Associative Memory begins.
00:08:04  0x60 (13)  All ROM items comparison with Associative Memory completed.
00:08:04  0x60 (14)  All ROM items comparison with Associative Memory produced correct results.
```

(b) GUI to associatively search ROM items against Associative Memory.

Figure 6-10: The FrontPanel testing GUI to read-out and associatively compare Item Memory hyper-vectors against the HDC processor's Associative Memory. (a) ROM and Continous Item Memory hyper-vectors are read out and matched to expected vectors for each Item Memory address. (b) Each ROM item is associatively searched against 32 random hyper-vector loaded into the Associative Memory; all 1024 items produced the expected closest match and hamming distance output.



TestLanguageVectors: Test encoding of 64 random test vectors from EUROPARL Language Recognition

CLK (kHz) 4800

Inspection of Encoder configuration

useEnoderEMG? EMGop[0] 0 EMGop[1] 0 muxA=EMG[0]? muxB=EMG[0]?

useEncoder2? EMG encoding? Skip Acc? A[0] 0 B[0] 0 op[0] 0

A[1] 0 B[1] 1 op[1] 3 A[2] 0 B[2] 2 op[2] 3 A[3] 3 B[3] 3 op[3] 0

A[4] 4 B[4] 4 op[4] 0 A[5] 5 B[5] 5 op[5] 0 A[6] 6 B[6] 6 op[6] 0

Results: Classes match? Vectors match? # vectors mismatch: 0 Total D(ham): 0

PMIC Standby: Vavg= 5168 2547824

PMIC Measurements: Measure Vavg= 5159 Test Pavg= 2840631 # CLK cycles= 27142

```
Time      Source      Message
09:08:07  0x60 (21)   The PMIC is configured for measurement.
09:08:13  0x60 (4)    The load for all language class vectors begins.
09:08:13  0x60 (5)    The load for all language class vectors completed successfu
09:08:13  0x60 (7)    The reading for all loaded language class vectors begins.
09:08:13  0x60 (8)    The reading for all loaded language class vectors complete..
09:08:13  0x60 (9)    The read vectors of all loaded language classes matched pe.
09:08:14  0x60 (17)   Encoder configuration for 3gram begins.
09:08:14  0x60 (20)   Encoder configuration completed successfully.
09:08:14  0x60 (18)   Encoder configuration read begins.
09:08:14  0x60 (20)   Encoder configuration completed successfully.
09:08:16  0x60 (12)   The encoding of EUROPARL Language tests begins.
09:08:16  0x60 (13)   The encoding of EUROPARL Language tests completed suc.
09:08:16  0x60 (14)   The encoded vectors of EUROPARL Language tests matche..
09:08:16  0x60 (23)   The PMIC standby VDD and Vshunt measurement ends.
```

V= 5155 I= 498

(a) GUI after testing encoded hyper-vectors for EUROPARL language recognition.

TestEMGVectors: Test encoding of 64 random test vectors from EMG2 hand-gesture recognition (Subject 2, Session 7, ...

CLK (kHz) 20161

Inspection of Encoder configuration

useEnoderEMG? EMGop[0] 0 EMGop[1] 0 muxA=EMG[0]? muxB=EMG[0]?

useEncoder2? EMG encoding? Skip Acc? A[0] 0 B[0] 0 op[0] 0

A[1] 0 B[1] 1 op[1] 3 A[2] 0 B[2] 2 op[2] 3 A[3] 0 B[3] 3 op[3] 3

A[4] 0 B[4] 4 op[4] 3 A[5] 5 B[5] 5 op[5] 0 A[6] 6 B[6] 6 op[6] 0

Results: Classes match? Vectors match? # vectors mismatch: 0 Total D(ham): 0

PMIC Standby: Vavg= 5156 22979638

PMIC Measurements: Measure Vavg= 5154 Test Pavg= 34726479 # CLK cycles= 58752

```
Time      Source      Message
16:05:51  0x60 (21)   The PMIC is configured for measurement.
16:05:53  0x60 (21)   The PMIC is configured for measurement.
16:05:54  0x60 (21)   The PMIC is configured for measurement.
16:05:57  0x60 (21)   The PMIC is configured for measurement.
16:05:58  0x60 (21)   The PMIC is configured for measurement.
16:05:59  0x60 (4)    The load for all class vectors begins.
16:05:59  0x60 (5)    The load for all class vectors completed successfully.
16:06:00  0x60 (7)    The reading for all loaded class vectors begins.
16:06:00  0x60 (8)    The reading for all loaded class vectors completed successf.
16:06:00  0x60 (9)    The read vectors of all loaded classes matched perfectly.
16:06:01  0x60 (17)   Encoder configuration for 3gram begins.
16:06:01  0x60 (20)   Encoder configuration completed successfully.
16:06:01  0x60 (18)   Encoder configuration read begins.
16:06:01  0x60 (20)   Encoder configuration completed successfully.
```

V= 5160 I= 4465

(b) GUI after testing encoded hyper-vectors fpr EMG hand-gesture recognition.

Figure 6-11: Encoder tests for Lanuage and EMG hand-gesture recognition
Tests are encoded and outputted: read out hyper-vector are matched with correct hyper-vectors. (a) 64 random test sentences from the EUROPARL corpus [90] are entered into the chip for encoding and reading out. (b) 64 random test sentences from the EMG hand-gesture recognition data set (from [39]) for subject 2, session 7, trial 1 are entered into the chip for encoding and reading out.
Note the (averaged) PMIC measurements of VDD voltage and power when the chip is idle or testing, reported as multiples of 195.3125 μV and 813.802 pW respectively.

testing of Encoder. The mode `READ_ENCODER` is used to read out the encoded hyper-vector at a rate of 1 byte/cycle in little-endian order after the encoding completes.

All 64 **tests of EUROPARL language recognition and EMG hand-gesture recognition returned *perfectly correct* encoded hyper-vectors.** This confirms the correct functionality of Encoder for the 2 applications listed in benchmark table 6.1 for measurements.

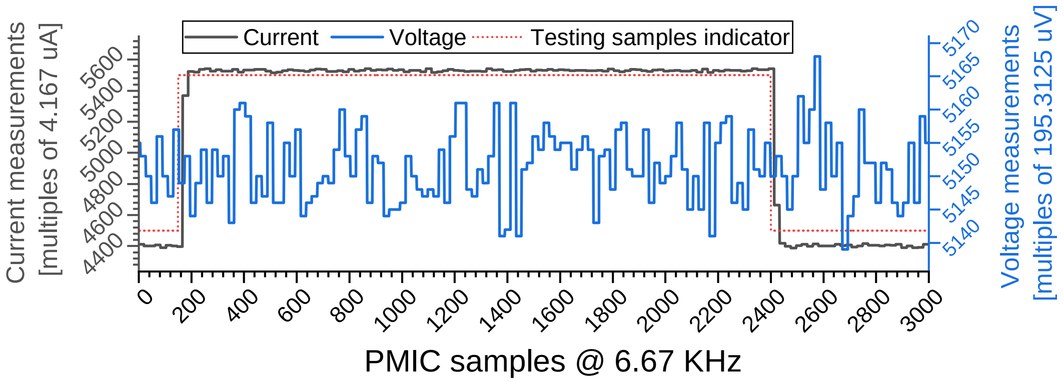


Figure 6-12: A measurement trace from PMIC
The dotted red line is an indicator flag output from the FPGA test-bench to mark the duration when the tests are ongoing in the HDC processor chip.

6.3 Inference energy measurements on chip

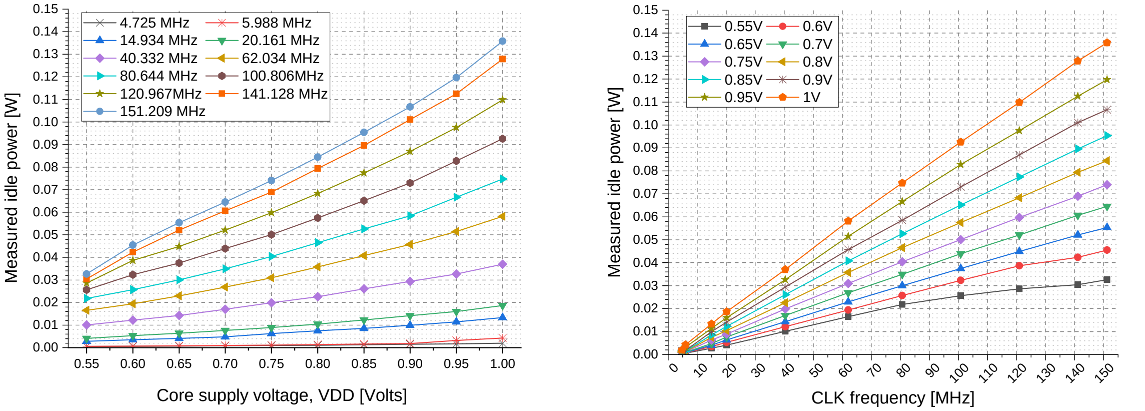
After the functionality of the three major components of the HDC processor chip has been verified in the previous section, the inference energy and power measurements from on-board PMIC is described in this section.

Figure 6-12 shows an example trace of VDD voltage and current measurements for the cope supply VDD. *Unfortunately, corresponding measurements for I/O voltage supply VDDPST could not be obtained as the PCB's connection to PMIC B (see figure 6-5(a)) was faulty.*

Note the elevated current value but no appreciable change in average VDD voltage while tests are ongoing – this indicates a much higher testing power consumption than during the chip's idle state. PMIC measurement traces, such as shown in figure 6-12, are used to calculate average VDD voltage and power consumption values over some idle time (to estimate idle power) and while testing. Figure 6-11 illustrates how the final PMIC measurements are displayed on the FrontPanel GUI for the 2 applications of the benchmark table 6.1 tested on chip.

In the chip's idle state, all inputs except `CLK` are held constant – only the clock input `CLK` transitions periodically. Therefore, the clock net and all associated nets are toggling – thereby contributing a large dynamic component to the chip's idle power consumption. The power consumed by the HDC processor in the idle state for a collection of VDD and `CLK` frequencies are plotted in figure 6-13. Indeed, as shown in figure 6-14, the measured leakage power consumption (when all chip inputs *including* `CLK` are held constant) was far smaller than the idle power consumption at the same VDD core supply voltages. The measured leakage power ≤ 0.8 mW was found to be a relatively small value for the chip – indicating an adequately high threshold voltage for most of its field-effect transistors.

On using linear predictors for chip power consumption to estimate energy/prediction: As one may expect in digital circuits, the idle power varies approx-



(a) Idle power vs. VDD at various `CLK` frequencies.

(b) Idle power vs. `CLK` frequency for various VDD voltages.

Figure 6-13: Idle power measurements.

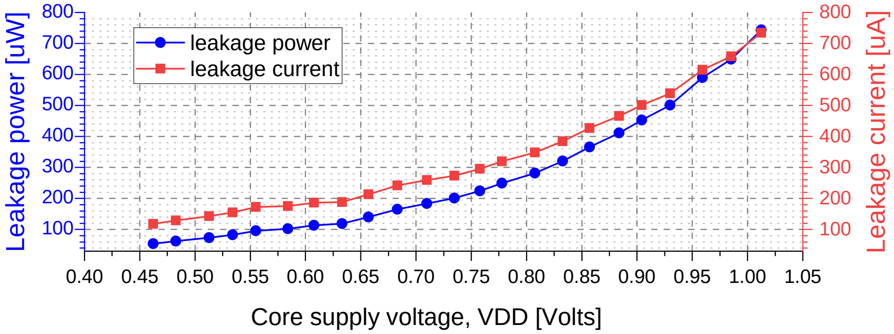
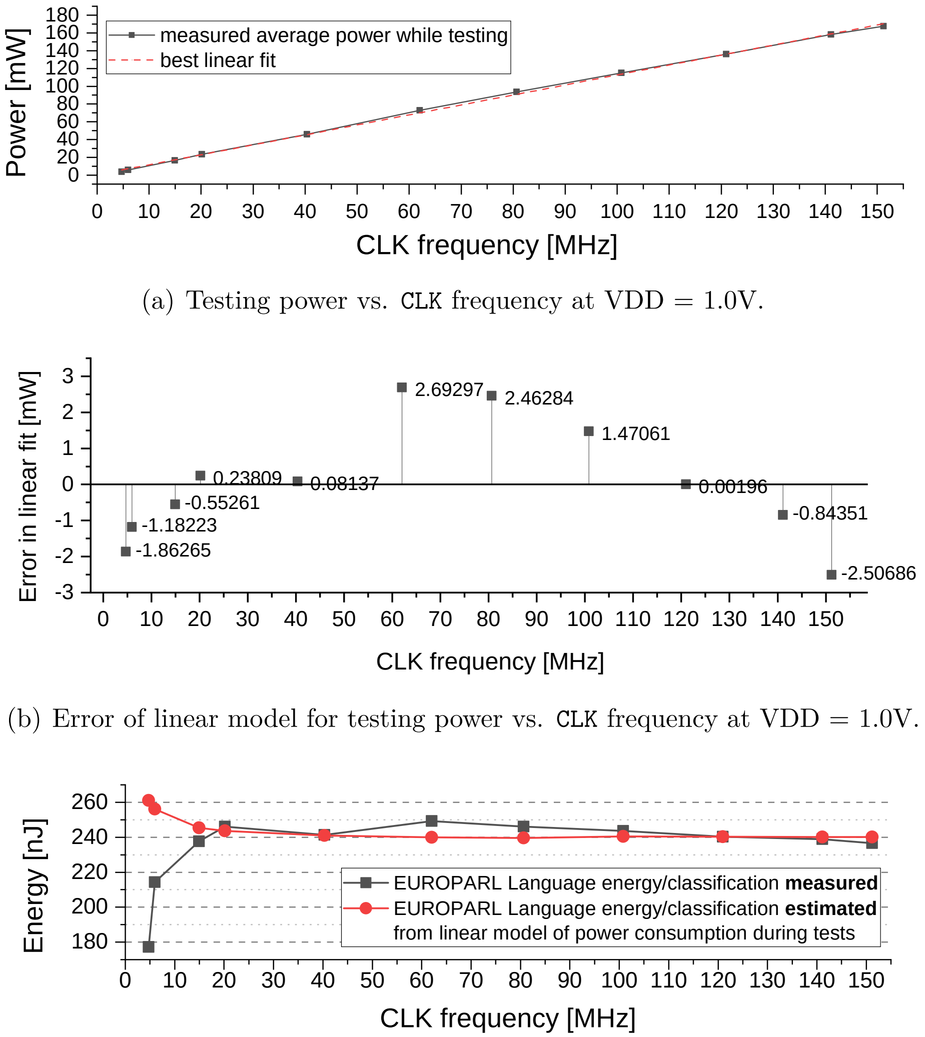


Figure 6-14: Measurements of leakage power at various VDD voltages. Leakage power is measured when chip inputs including `CLK` are held constant.

imately quadratically with supply voltage VDD at constant `CLK` frequencies (figure 6-13(a)) and approximately linearly with `CLK` frequency at constant VDD voltage (figure 6-13(b)). The linear dependence of power consumed with `CLK` frequency was also found to be a great model for estimating power consumed when testing applications on chip, such as EUROPARL language recognition at VDD = 1.0V as plotted in 6-15(a) However, even small errors in the power consumed at *low* `CLK` frequencies can lead to large discrepancies between estimated and measured energy/classification. Indeed, while $f = 4.7$ and $f = 150$ MHz lead to ≈ -2 mW error in estimated testing power (figure 6-15(b)), the estimation error in energy/classification for $f = 4.7$ MHz is about 37× that for $f = 150$ MHz as shown in figure 6-15(c). This is in contrast with the fact that since PMIC samples at a constant rate of ≈ 6670 samples/s, the identical test program at a lower `CLK` frequency takes longer, yields proportioanlly more samples and therefore results in a more accurate (averaged) VDD voltage and power measurements.



(a) Testing power vs. `CLK` frequency at VDD = 1.0V.

(b) Error of linear model for testing power vs. `CLK` frequency at VDD = 1.0V.

(c) Error of energy/classification estimated by the linear model for testing power vs. `CLK` frequency at VDD = 1.0V.

Figure 6-15: Using the linear model for power consumption leads to inaccurate energy/predioction estimates at low frequencies.
(a) The best fitting line is $(1.1223f + 0.5042)$ mW for the test power consumption of language recognition at `CLK` frequency of f MHz and VDD = 1.0 Volts. (b) The best fitting line is a great predictor for the testing power consumed. (c) Since energy/classification $\propto f^{-1}$, power estimation errors by the linear power model leads to *large discrepancies* in estimates of energy/prediction at low `CLK` frequencies.

6.3.1 Measured inference energy for Language Recognition

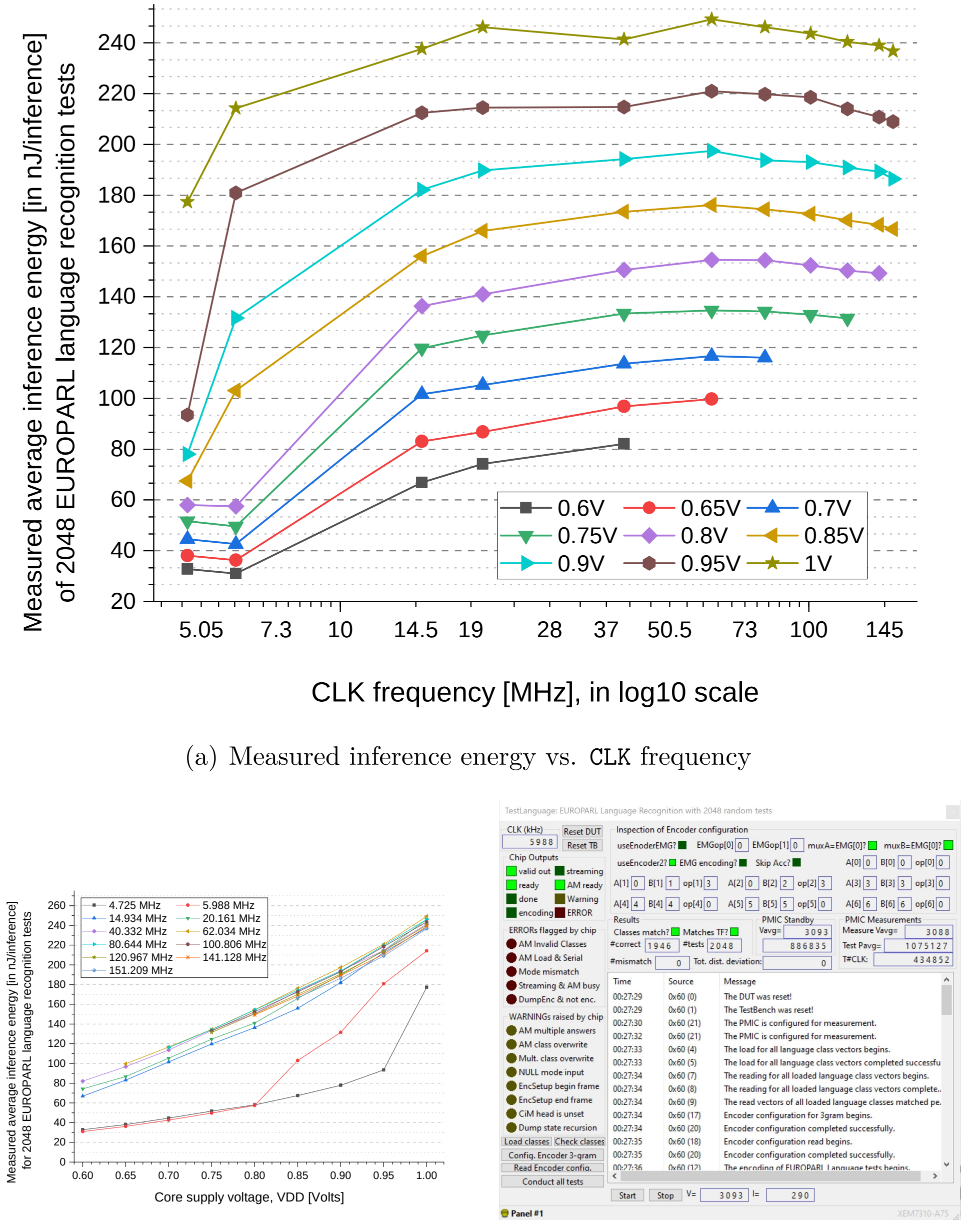
Language recognition for 21 European languages [65] were conducted on the 28nm HDC processor chip. After loading in the language vectors (already pre-trained offline) into the chip's Associative Memory, 2048 random sentences were administered for testing; all 2048 tests were repeated 500 times *without break* to collect a large enough sample of PMIC measurements. A total of $434852 \times 500 \approx 2.174 \times 10^8$ `CLK` cycles were spent continuously for testing.

The chip's outputs for each test in every repetition were compared with that of the offline software simulation. Such testing for EUROPARL language recognition was repeated for multiple `CLK` frequencies and VDD voltages, and the measured energy/classification is reported *if and only if* the chip produced the expected outputs (i.e. closest language's Associative Memory address and its hamming distance to the test's encoded hyper-vector) *for all* 2048 *tests during each of the* 500 *repetitions*.

Figure 6-16 plots the measured energy/classification from the testing data collected. Measurements show that one can reliably conduct language recognition for less than 40 nJ/classification; specifically 32.79 ± 0.36 nJ for VDD ≈ 0.6V and $f = 4.725$ MHz, 38.12 ± 0.36 nJ for VDD ≈ 0.65V and $f = 4.725$ MHz, 31.02 ± 0.36 nJ for VDD ≈ 0.6V and $f = 5.988$ MHz, and 36.26 ± 0.36 nJ for VDD ≈ 0.65V, $f = 5.988$ MHz.

Table 6.3 compares the best measurement with previous literature on HDC data-paths for language recognition. The comparison reveals interesting caveats:

1. The lowest energy/classification measured on chip is also the smallest reported on table 6.3, with a competitive overall classification accuracy.
2. JSSC 2018 [136] implemented only 32 bits of the HDC dimension, multiplexing the entire data-path through the chip when testing. It used Carbon-Nanotube FETs (CNFETs) for logic, integrated with Resistive RAM (RRAM) memories; and on-chip device variability to produce item hyper-vectors (instead of ROMs on this chip). However, they report *far greater* average inference energy cost while offering a similar order of prediction throughput.
3. TCAS 2021 [174] used digital standard cells to implement its data-path, like this chip. However, it was designed specifically to reduce leakage power consumption i.e. for a lower core supply and `CLK` frequency. Therefore, it has *far smaller prediction throughput*. However, the average inference energy cost is about an order of magnitude larger.
4. Nature Electronics 2020 [70] offers an interesting comparison. They use 90nm post-synthesis estimates for logic at a higher core supply voltage of 1.2V and on-chip 90nm Phase-Change Memory (PCM), implementing the Associative Memory and part of the Encoder, with only 0.1V supply. Even with a greater HDC dimension $d = 10,000$ bits and a *far higher prediction throughput* of 0.9 million preds/s, they report only 430 nJ/classification. An *estimate* for their design for $d = 2048$ would be $0.2 \times 430 = 86$ nJ/classification. The analogous measurement on this chip is 236.61 ± 0.36 nJ/classification for VDD = 1.0V and $f = 151.209$ MHz – i.e. about 2.75× greater energy cost! This achieves the highest prediction throughput of 712141 preds/s for this chip but is still $\approx 20\%$ smaller than their throughput.



(a) Measured inference energy vs. CLK frequency

TestLanguage: EUROPARL Language Recognition with 2048 random tests

CLK (kHz) 5988

Chip Outputs: 🟩 valid out ■ streaming 🟩 ready 🟩 AM ready ■ done ■ Warning ■ encoding ■ ERROR

ERRORs flagged by chip: ● AM Invalid Classes ● AM Load & Serial ● Mode mismatch ● Streaming & AM busy ● DumpEnc & not enc.

WARNINGs raised by chip: ● AM multiple answers ● AM class overwrite ● Mult. class overwrite ● NULL mode input ● EncSetup begin frame ● EncSetup end frame ● CiM head is unset ● Dump state recursion

Inspection of Encoder configuration

useEnoderEMG? ■ EMGop[0] 0 EMGop[1] 0 muxA=EMG[0]? 🟩 muxB=EMG[0]? 🟩

useEncoder2? 🟩 EMG encoding? ■ Skip Acc? ■ A[0] 0 B[0] 0 op[0] 0

A[1] 0 B[1] 1 op[1] 3 A[2] 0 B[2] 2 op[2] 3 A[3] 3 B[3] 3 op[3] 0

A[4] 4 B[4] 4 op[4] 0 A[5] 5 B[5] 5 op[5] 0 A[6] 6 B[6] 6 op[6] 0

Results: Classes match? 🟩 Matches TF? 🟩 #correct 1946 #tests 2048 #mismatch 0 Tot. dist. deviation: 0

PMIC Standby: Vavg= 3093 886835

PMIC Measurements: Measure Vavg= 3088 Test Pavg= 1075127 T#CLK: 434852

```
Time       Source      Message
00:27:29   0x60 (0)    The DUT was reset!
00:27:29   0x60 (1)    The TestBench was reset!
00:27:30   0x60 (21)   The PMIC is configured for measurement.
00:27:32   0x60 (21)   The PMIC is configured for measurement.
00:27:33   0x60 (4)    The load for all language class vectors begins.
00:27:33   0x60 (5)    The load for all language class vectors completed successfu
00:27:34   0x60 (7)    The reading for all loaded language class vectors begins.
00:27:34   0x60 (8)    The reading for all loaded language class vectors complete..
00:27:34   0x60 (9)    The read vectors of all loaded language classes matched pe.
00:27:34   0x60 (17)   Encoder configuration for 3gram begins.
00:27:34   0x60 (20)   Encoder configuration completed successfully.
00:27:35   0x60 (18)   Encoder configuration read begins.
00:27:35   0x60 (20)   Encoder configuration completed successfully.
00:27:36   0x60 (12)   The encoding of EUROPARL Language tests begins.
```

V= 3093 I= 290

Panel #1

(b) Measured inference energy vs. VDD

(c) Measurement at VDD ≈ 0.6V, $f \approx 6$ MHz

Figure 6-16: Measured energy/prediction for 2048 random tests of EUROPARL language recognition.

Publication:	JSSC 2018 [136]	TCAS I, 2021 [174]	Nature Electronics 2020 [70]	**This work**
Data set	2 languages *only* from EUROPARL [90]	EUROPARL [90]	Wortschatz [175] for training EUROPARL [90] for testing	EUROPARL [90]
HDC dim.	2048 bits	2048 bits	10000 bits	2048 bits
Accuracy	$\approx 96\%$	$\approx 84\%$	96.99%	94.44%
Platform	ASIC with RRAM memory, CNFET logic	ASIC with 22nm HK/MG CMOS	ASIC with 90nm PCM memory and 65nm CMOS logic	ASIC with 28nm HK/MG CMOS
Data-path width	32 bits	2048 bits	10000 bits	2048 bits
Meas. type	directly from chip	simulated post-layout	partly measured from chip$^{(1)}$	directly from chip
Core supply	3.0 V	0.6 V	1.2 V (logic), 0.1 V (PCM memory)	0.6 V
Throughput	13126 preds/s	72 preds/s	$\approx 9 \times 10^5$ preds/s	28201 preds/s
Pred. latency	76.2 μs	14 ms	1.2 μs	35.5 μs
Energy/pred.	411.4 nJ	332 nJ	430 nJ	31.02 ± 0.36 nJ

Table 6.3: Comparison of measured energy per inference for Language Recognition. The reported energy/classification are for VDD = 0.6 V and CLK frequency is 5.988 MHz. $^{(1)}$The Phase-Change Memory (PCM) measurements in [70], implementing the Associative Memory and 2-minterm approximation (for binding n-grams during encoding), were done directly on chip – rest of its data-path was simulated post-synthesis in 65nm CMOS.

6.3.2 Inference energy for EMG hand-gesture recognition

EMG hand-gesture recognition implemented in [38] and [39] have identical encoding algorithm – a channel-value product in the first encoding stage (called spatial encoding) followed by a 5-gram in the second encoding stage (called temporal encoding). The data set in [38] contains 5 gestures (4 single degrees-of-freedom gesture and a "rest" gesture) obtained from 3 subjects over 3 sessions. To save testing time, only the first session for each subject was considered for on-chip measurements (as tabulated in table 6.1). In [39], an augmented data set containing a total of 21 gestures (12 single degrees-of-freedom gestures, 8 multiple degrees-of-freedom gestures and a "rest" gesture) obtained from 5 subjects over 8 recording sessions, each further composed of 5 trials. While a large number of experiments were conducted in [39], only experiments 1 and 2 of [39] were performed for on-chip measurements (see table 6.1).

Unlike transliterated test sentences for language recognition which are of variable length, EMG test samples are of a *fixed length*: for the data sets [38, 39] considered here, a single test sample is a filtered sequence of 5 consecutive measurements over 64 channel-electrodes. Therefore, the number of cycles required to compute a prediction for EMG hand-gesture recognition is also fixed. However, there are two methods in which gesture classification may be conducted. This is because EMG gestures are modeled as *a single n-gram* of spatially-encoded vectors, unlike a *superposition* of n-grams for language recognition. The two methods of conducting hand-gesture prediction on the test sample are as follows:

- **Testing each n-gram separately.** For the test sample composed of a time-series of filtered, 64-channel measurements denoted by a sequence of 64-vectors $\vec{v}^{(1)}, \vec{v}^{(2)} \ldots \vec{v}^{(k)}$, the corresponding sequence of 5-grams

  $$(\vec{v}^{(1)}, \vec{v}^{(2)} \ldots \vec{v}^{(5)}), (\vec{v}^{(2)}, \vec{v}^{(3)} \ldots \vec{v}^{(6)}), \ldots, (\vec{v}^{(k-4)}, \vec{v}^{(k-3)} \ldots \vec{v}^{(k)})$$

  are inputted to the chip one-at-a-time for testing. Following the development of two-stage valid signalling in section 4.2, the 5-gram $(\vec{v}^{(1)}, \vec{v}^{(2)} \ldots \vec{v}^{(5)})$ is inputted as the following sequence of valids and inputs (denoted `valid`$\langle$`item`$\rangle$):

  $\texttt{V}\langle\text{Channel 1}\rangle, \texttt{V}\langle v_1^{(1)}\rangle, \texttt{V}\langle\text{Channel 2}\rangle, \texttt{V}\langle v_2^{(1)}\rangle, \ldots \texttt{V}\langle\text{Channel 64}\rangle, \texttt{V}\langle v_{64}^{(1)}\rangle, \texttt{D}\langle\text{don't care}\rangle$

  $\texttt{V}\langle\text{Channel 1}\rangle, \texttt{V}\langle v_1^{(2)}\rangle, \texttt{V}\langle\text{Channel 2}\rangle, \texttt{V}\langle v_2^{(2)}\rangle, \ldots \texttt{V}\langle\text{Channel 64}\rangle, \texttt{V}\langle v_{64}^{(2)}\rangle, \texttt{D}\langle\text{don't care}\rangle$

  $\cdots$ *similar inputs for* $\vec{v}^{(3)}$ *and* $\vec{v}^{(4)}$ $\cdots$

  $\texttt{V}\langle\text{Channel 1}\rangle, \texttt{V}\langle v_1^{(5)}\rangle, \texttt{V}\langle\text{Channel 2}\rangle, \texttt{V}\langle v_2^{(5)}\rangle, \ldots \texttt{V}\langle\text{Channel 64}\rangle, \texttt{V}\langle v_{64}^{(5)}\rangle, \texttt{D}\langle\text{don't care}\rangle$

  $\texttt{EE}\langle\text{don't care}\rangle$

  as illustrated in figure 4-12. Crucially, note the use of *global termination valid-signal* EE at the end, which instructs the processor to flush both encoding stage's pipelines and ends the entire computation. The computation is recommenced for the next 5-gram $(\vec{v}^{(2)}, \vec{v}^{(3)} \ldots \vec{v}^{(6)})$, and this is repeated for the remaining 5-grams.

  In this mode of testing, each prediction costs 707 cycles. EMG hand-gesture recognition for data sets [38, 39] were conducted on the 28nm HDC processor chip. After loading in the gesture vectors (already pre-trained offline) into the chip's Associative Memory, 512 random gesture 5-grams were administered for testing; all 512 tests were repeated 500 times *without break* to collect a large enough sample of PMIC measurements. A total of $361984 \times 500 \approx 1.81 \times 10^8$ `CLK` cycles were spent continuously for testing. The chip's outputs for each test in every repetition were compared with that of the offline software simulation. Such testing was repeated for multiple `CLK` frequencies and VDD voltages, and the measured energy/classification is reported *if and only if* the chip produced the expected outputs (i.e. closest gesture's Associative Memory address and its hamming distance to the test's encoded hyper-vector) *for all* 512 *tests during each of the* 500 *repetitions*.

Figure 6-18 plots the measured energy/classification from the testing data collected. Measurements show that one can reliably conduct gesture recognition, each 5-gram at a time, for less than 200 nJ/classification; specifically 144.52 ± 1.22 nJ for VDD ≈ 0.6V and $f = 4.725$ MHz, 167.44 ± 1.22 nJ for VDD ≈ 0.65V and $f = 4.725$ MHz, 196.09 ± 1.22 nJ for VDD ≈ 0.7V and $f = 4.725$ MHz, 134.43 ± 1.22 nJ for VDD ≈ 0.6V and $f = 5.988$ MHz and 165.07 ± 1.22 nJ for VDD ≈ 0.65V, $f = 5.988$ MHz.

Inputting each 5-gram separately is beneficial for testing the chip and for suppressing n-grams during transitions between hand-gestures. However, this does not use the advantage of pipelining afforded by HLU layers of the second encoding stage (refer section 4.1.2). Pipelining improves the throughput and the energy cost for each inference.

- **Streaming samples and pipelining n-grams.** This mode of testing uses HLU layers of the Encoder's second stage to pipeline the sequence of n-grams. The *entire* test time-series of filtered, 64-channel measurements denoted by a sequence of 64-vectors $\vec{v}^{(1)}, \vec{v}^{(2)} \ldots \vec{v}^{(k)}$ are *inputted sequentially* so that the corresponding sequence of 5-grams

  $$(\vec{v}^{(1)}, \vec{v}^{(2)} \ldots \vec{v}^{(5)}), (\vec{v}^{(2)}, \vec{v}^{(3)} \ldots \vec{v}^{(6)}), \ldots, (\vec{v}^{(k-4)}, \vec{v}^{(k-3)} \ldots \vec{v}^{(k)})$$

  are generated within the chip's Encoder during testing. The sequence of valid-input pairs, denoted $\texttt{valid}\langle\texttt{item}\rangle$, are streamed into the chip as follows:

  $\texttt{V}\langle\text{Channel 1}\rangle, \texttt{V}\langle v_1^{(1)}\rangle, \texttt{V}\langle\text{Channel 2}\rangle, \texttt{V}\langle v_2^{(1)}\rangle, \ldots \texttt{V}\langle\text{Channel 64}\rangle, \texttt{V}\langle v_{64}^{(1)}\rangle, \texttt{D}\langle\text{don't care}\rangle$

  $\texttt{V}\langle\text{Channel 1}\rangle, \texttt{V}\langle v_1^{(2)}\rangle, \texttt{V}\langle\text{Channel 2}\rangle, \texttt{V}\langle v_2^{(2)}\rangle, \ldots \texttt{V}\langle\text{Channel 64}\rangle, \texttt{V}\langle v_{64}^{(2)}\rangle, \texttt{D}\langle\text{don't care}\rangle$

  $\cdots$ *similar inputs for* $\vec{v}^{(3)}$ *and* $\vec{v}^{(4)}$ $\cdots$

  $\texttt{V}\langle\text{Channel 1}\rangle, \texttt{V}\langle v_1^{(5)}\rangle, \texttt{V}\langle\text{Channel 2}\rangle, \texttt{V}\langle v_2^{(5)}\rangle, \ldots \texttt{V}\langle\text{Channel 64}\rangle, \texttt{V}\langle v_{64}^{(5)}\rangle, \texttt{D}\langle\text{don't care}\rangle$

  $\texttt{V}\langle\text{Channel 1}\rangle, \texttt{V}\langle v_1^{(6)}\rangle, \texttt{V}\langle\text{Channel 2}\rangle, \texttt{V}\langle v_2^{(6)}\rangle, \ldots \texttt{V}\langle\text{Channel 64}\rangle, \texttt{V}\langle v_{64}^{(6)}\rangle, \texttt{D}\langle\text{don't care}\rangle$

  $\cdots\cdots\cdots$

  $\texttt{V}\langle\text{Channel 1}\rangle, \texttt{V}\langle v_1^{(k)}\rangle, \texttt{V}\langle\text{Channel 2}\rangle, \texttt{V}\langle v_2^{(k)}\rangle, \ldots \texttt{V}\langle\text{Channel 64}\rangle, \texttt{V}\langle v_{64}^{(k)}\rangle, \texttt{D}\langle\text{don't care}\rangle$

  $\texttt{EE}\langle\text{don't care}\rangle$

  Crucially, the *global termination valid-signal* EE is used only when the entire test sequence is completely consumed by the chip. Using the STREAM mode of operation, the second layer of Encoder is instructed to *output instead of accumulating* all Valid (V) hyper-vectors. In this mode, the Associative Memory is also enabled while the Encoder is encoding inputs, commencing on an associative search whenever the Encoder's second stage outputs a valid encoded hyper-vector. In this manner, the chip produces a sequence of predicted gesture (Associative Memory) labels and the test hyper-vector's hamming distance to it while the inputs are streamed in and processed.
  In this mode of testing, a prediction is reported every 136 CLK cycles. EMG hand-gesture recognition for data sets [38, 39] were conducted on the 28nm HDC processor chip. After loading in the gesture vectors (already pre-trained offline) into the chip's Associative Memory, the test trace was *streamed in* for testing, and the entire trace was repeated 500 times *without break* to collect a large enough sample of PMIC measurements. A total of $261096 \times 500 \approx 1.31 \times 10^8$ CLK cycles were spent continuously for testing. The chip's outputs for each *non-transition* 5-gram contained in the test trace, in every repetition, were compared with that of the offline software simulation. This routine was repeated for multiple CLK frequencies and VDD voltages, and the measured energy/classification is reported *if and only if* the chip produced the expected outputs (i.e. closest gesture's Associative Memory address and its hamming distance to the test's encoded hyper-vector) *for all non-transition* 5*-grams during each of the* 500 *repetitions.*

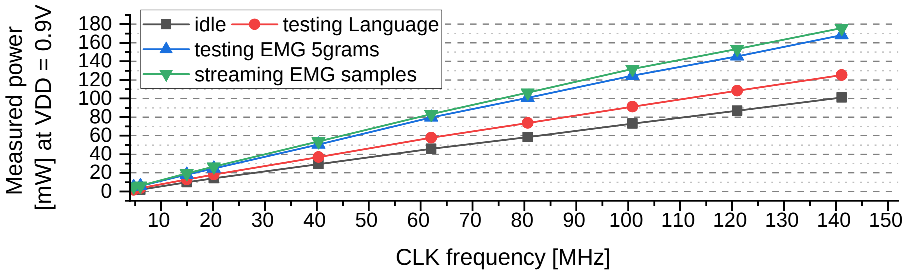


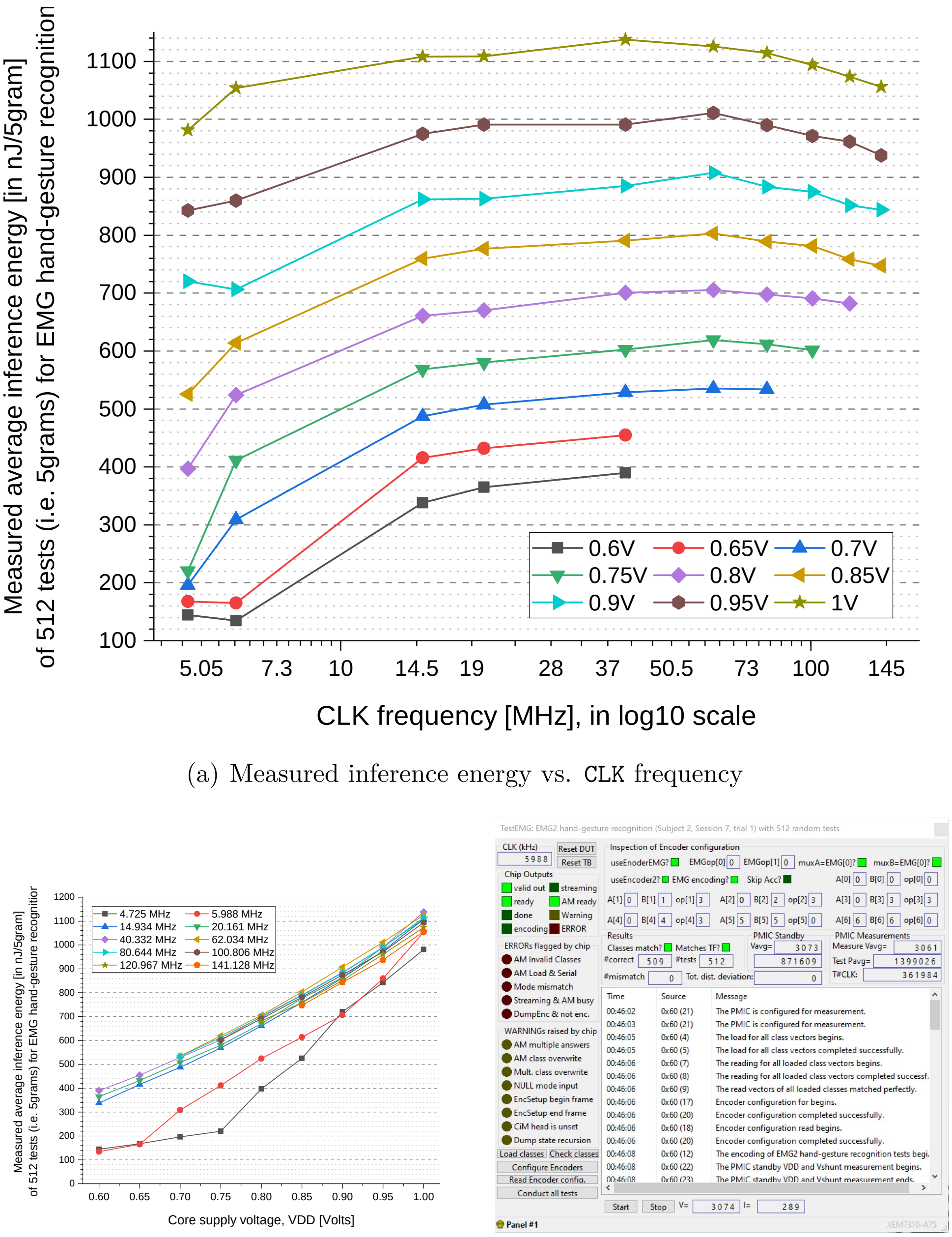
Figure 6-17: Measured test power for Language recognition and EMG hand-gesture recognition at VDD ≈ 1V.

Figure 6-19 plots the measured energy/classification from the testing data collected. Measurements show that one can reliably conduct gesture recognition, for less than 40 nJ/classification; specifically 29.28 ± 0.24 nJ for VDD ≈ 0.6V and $f = 4.725$ MHz, 34.41 ± 0.24 nJ for VDD ≈ 0.65V and $f = 4.725$ MHz, 39.29 ± 0.24 nJ for VDD ≈ 0.7V and $f = 4.725$ MHz and 28.59 ± 0.24 nJ for VDD ≈ 0.6V, $f = 5.988$ MHz.

EMG gesture classifications consume greater power on chip than language recognition, as shown in figure 6-17 for VDD ≈ 1V. EMG streaming tests consume greater power than without since the associative search happens simultaneously with encoding.

Since streaming inputs leads to *real-time gesture recognition* – a property of most *competitive* EMG hand-gesture recognition systems [39, 38, 87] – only the best measurements for the chip's streaming tests are considered for comparison with the available literature. Comparisons are available in table 6.4. Note the following:

1. There is a diversity of number of channels and number of gestures in the EMG hand-gesture data sets mentioned in table 6.4. All except [39] use gestures with a single degree-of-freedom; they are considered to be easier to classify and result in comparatively higher classification accuracy. This chip was verified with both single-degree-of-freedom gestures (data set from [38]) and multiple-degrees-of-freedom gestures (data set from [39]). Since energy cost per inference increases with larger number of channels, the inference energy per channel is considered as the final metric of efficiency.
2. Streaming in tests increases the throughput of gestures predicted by the HDC processor chip due to pipelining, but does not change the latency of processing a gesture. Therefore, even though a gesture label is produced every 136 CLK cycles, each gesture still needs 707 cycles to be classified by the chip. The resultant prediction throughput and latency for the reported best energy/prediction measurement at VDD ≈ 6.0 V and $f = 5.988$ MHz are shown in table 6.4.
3. The left-most three columns of table 6.4 for [176, 177, 178] implements EMG hand-gesture recognition in 32-bit micro-controllers using different algorithms. Significantly, [178] employs HDC on a 8-core, ultra-low-power micro-controller executing an optimized assembly program. Measurements confirm that ASIC implementation results in at least 3 orders of magnitude greater efficiency than micro-controllers, similar to CPUs as seen section 3.1.3.



(a) Measured inference energy vs. `CLK` frequency

(b) Measured inference energy vs. VDD

TestEMG: EMG2 hand-gesture recognition (Subject 2, Session 7, trial 1) with 512 random tests

CLK (kHz) 5988

Inspection of Encoder configuration

useEnoderEMG? 🟩 EMGop[0] 0 EMGop[1] 0 muxA=EMG[0]? 🟩 muxB=EMG[0]? 🟩

useEncoder2? 🟩 EMG encoding? 🟩 Skip Acc? 🟩

A[0] 0 B[0] 0 op[0] 0

A[1] 0 B[1] 1 op[1] 3 A[2] 0 B[2] 2 op[2] 3 A[3] 0 B[3] 3 op[3] 3

A[4] 0 B[4] 4 op[4] 3 A[5] 5 B[5] 5 op[5] 0 A[6] 6 B[6] 6 op[6] 0

Results

Classes match? 🟩 Matches TF? 🟩

#correct 509 #tests 512

#mismatch 0 Tot. dist. deviation: 0

PMIC Standby

Vavg= 3073

871609

PMIC Measurements

Measure Vavg= 3061

Test Pavg= 1399026

T#CLK: 361984

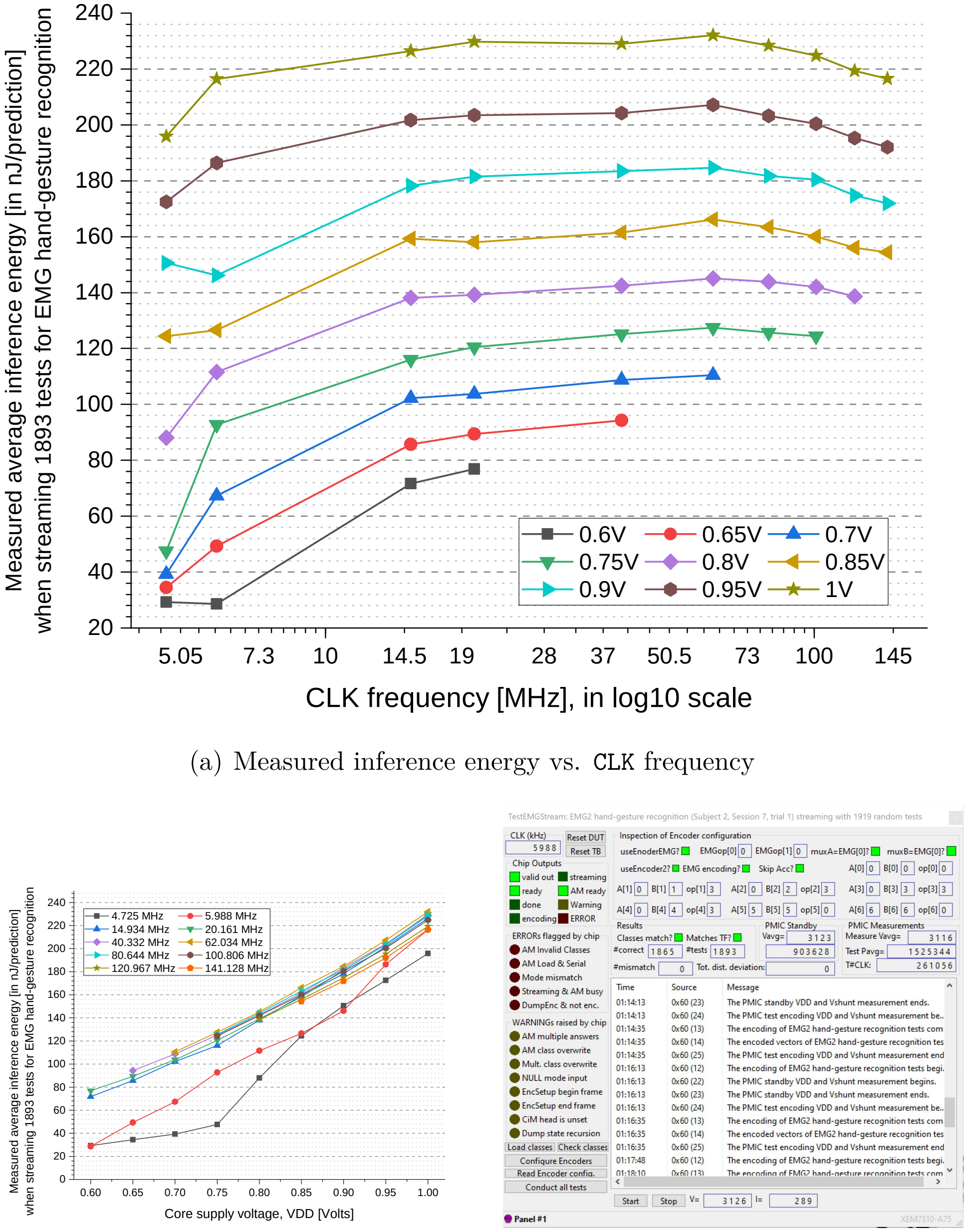
```
Time      Source     Message
00:46:02  0x60 (21)  The PMIC is configured for measurement.
00:46:03  0x60 (21)  The PMIC is configured for measurement.
00:46:05  0x60 (4)   The load for all class vectors begins.
00:46:05  0x60 (5)   The load for all class vectors completed successfully.
00:46:06  0x60 (7)   The reading for all loaded class vectors begins.
00:46:06  0x60 (8)   The reading for all loaded class vectors completed successf.
00:46:06  0x60 (9)   The read vectors of all loaded classes matched perfectly.
00:46:06  0x60 (17)  Encoder configuration for begins.
00:46:06  0x60 (20)  Encoder configuration completed successfully.
00:46:06  0x60 (18)  Encoder configuration read begins.
00:46:06  0x60 (20)  Encoder configuration completed successfully.
00:46:08  0x60 (12)  The encoding of EMG2 hand-gesture recognition tests begi.
00:46:08  0x60 (22)  The PMIC standby VDD and Vshunt measurement begins.
00:46:08  0x60 (23)  The PMIC standby VDD and Vshunt measurement ends.
```

V= 3074 I= 289

(c) Measurement at VDD ≈ 0.6V, $f \approx 6$ MHz for subject 2, session 7, trial 1 of data set [39]

Figure 6-18: Measured energy/prediction for 512 random tests of EMG hand-gesture recognition classified each 5-gram at a time.

EMG hand-gesture recognition for classifying each 5-gram at a time – without pipe-lining at the second encoding stage. Both data sets [38] and [39] require 707 cycles/prediction, and hence have same energy/inference.



(a) Measured inference energy vs. CLK frequency

(b) Measured inference energy vs. VDD

TestEMGStream: EMG2 hand-gesture recognition (Subject 2, Session 7, trial 1) streaming with 1919 random tests

CLK (kHz) 5988

Inspection of Encoder configuration
useEnoderEMG? 🟩 EMGop[0] 0 EMGop[1] 0 muxA=EMG[0]? 🟩 muxB=EMG[0]? 🟩
useEncoder2? 🟩 EMG encoding? 🟩 Skip Acc? 🟩 A[0] 0 B[0] 0 op[0] 0
A[1] 0 B[1] 1 op[1] 3 A[2] 0 B[2] 2 op[2] 3 A[3] 0 B[3] 3 op[3] 3
A[4] 0 B[4] 4 op[4] 3 A[5] 5 B[5] 5 op[5] 0 A[6] 6 B[6] 6 op[6] 0

Chip Outputs
🟩 valid out ■ streaming
🟩 ready 🟩 AM ready
■ done ■ Warning
■ encoding ■ ERROR

ERRORs flagged by chip
● AM Invalid Classes
● AM Load & Serial
● Mode mismatch
● Streaming & AM busy
● DumpEnc & not enc.

WARNINGs raised by chip
● AM multiple answers
● AM class overwrite
● Mult. class overwrite
● NULL mode input
● EncSetup begin frame
● EncSetup end frame
● CiM head is unset
● Dump state recursion

Results
Classes match? 🟩 Matches TF? 🟩
#correct 1865 #tests 1893
#mismatch 0 Tot. dist. deviation: 0

PMIC Standby
Vavg= 3123
903628

PMIC Measurements
Measure Vavg= 3116
Test Pavg= 1525344
T#CLK: 261056

Time	Source	Message
01:14:13	0x60 (23)	The PMIC standby VDD and Vshunt measurement ends.
01:14:13	0x60 (24)	The PMIC test encoding VDD and Vshunt measurement be..
01:14:35	0x60 (13)	The encoding of EMG2 hand-gesture recognition tests com
01:14:35	0x60 (14)	The encoded vectors of EMG2 hand-gesture recognition tes
01:14:35	0x60 (25)	The PMIC test encoding VDD and Vshunt measurement end
01:16:13	0x60 (12)	The encoding of EMG2 hand-gesture recognition tests begi.
01:16:13	0x60 (22)	The PMIC standby VDD and Vshunt measurement begins.
01:16:13	0x60 (23)	The PMIC standby VDD and Vshunt measurement ends.
01:16:13	0x60 (24)	The PMIC test encoding VDD and Vshunt measurement be..
01:16:35	0x60 (13)	The encoding of EMG2 hand-gesture recognition tests com
01:16:35	0x60 (14)	The encoded vectors of EMG2 hand-gesture recognition tes
01:16:35	0x60 (25)	The PMIC test encoding VDD and Vshunt measurement end
01:17:48	0x60 (12)	The encoding of EMG2 hand-gesture recognition tests begi.
01:18:10	0x60 (13)	The encoding of EMG2 hand-gesture recognition tests com

V= 3126 I= 289

(c) Measurement at VDD ≈ 0.6V, $f \approx 6$ MHz for subject 2, session 7, trial 1 of data set [39]

Figure 6-19: Measured energy/prediction for 1893 random tests of EMG hand-gesture recognition with samples streamed in continuously.

EMG hand-gesture recognition was performed by streaming in the entire measurement time-series, where the Encoder's pipelines produce 5-grams internally. Both data sets [38, 39] require 136 cycles/prediction and therefore have same energy/inference. In (c), a trace from trial 1, session 7 for subject 2 in the data set [39] was streamed in, containing 1919 5-grams of which 26 were during transitions. Inferences on the remaining 1893 gestures were considered.

4. FPGAs have far greater efficiency than micro-controllers, as shown by [39] – but still results in more than 50× energy cost compared to the chip.
5. Among the last 3 columns, this chip results in about $5 - 6\times$ greater energy efficiency per channel. In particular, it is *uniformly* better than [174] which uses the same data set [38] as this chip but has $\approx 300\times$ lower prediction throughput.
6. In [179], measurements from 90nm PCM memory were combined with 65nm post-synthesis logic simulations for EMG hand-gesture recognition on the data set [19]; as done for EUROPARL language recognition in [70]. The data set [19] is *far more rudimentary* compared to data sets [38, 39] tested on this chip: it has only 4 gestures and needs much higher HDC dimensions to get adequately high classification accuracy. However, their preliminary results show promise as the inference energy per channel is only 7.4× of this chip's. In fact, [179] can improved further as real-time EMG hand-gesture recognition could function with far lower HDC dimensions and prediction throughput than theirs.

6.3.3 Robustness of classification accuracy

While the previous sub-sections report the best measured inference energy costs when the chip's outputs match simulations *exactly*, it is instructive to measure the accuracy while the core supply VDD is reduced further to introduce timing errors in the chip.

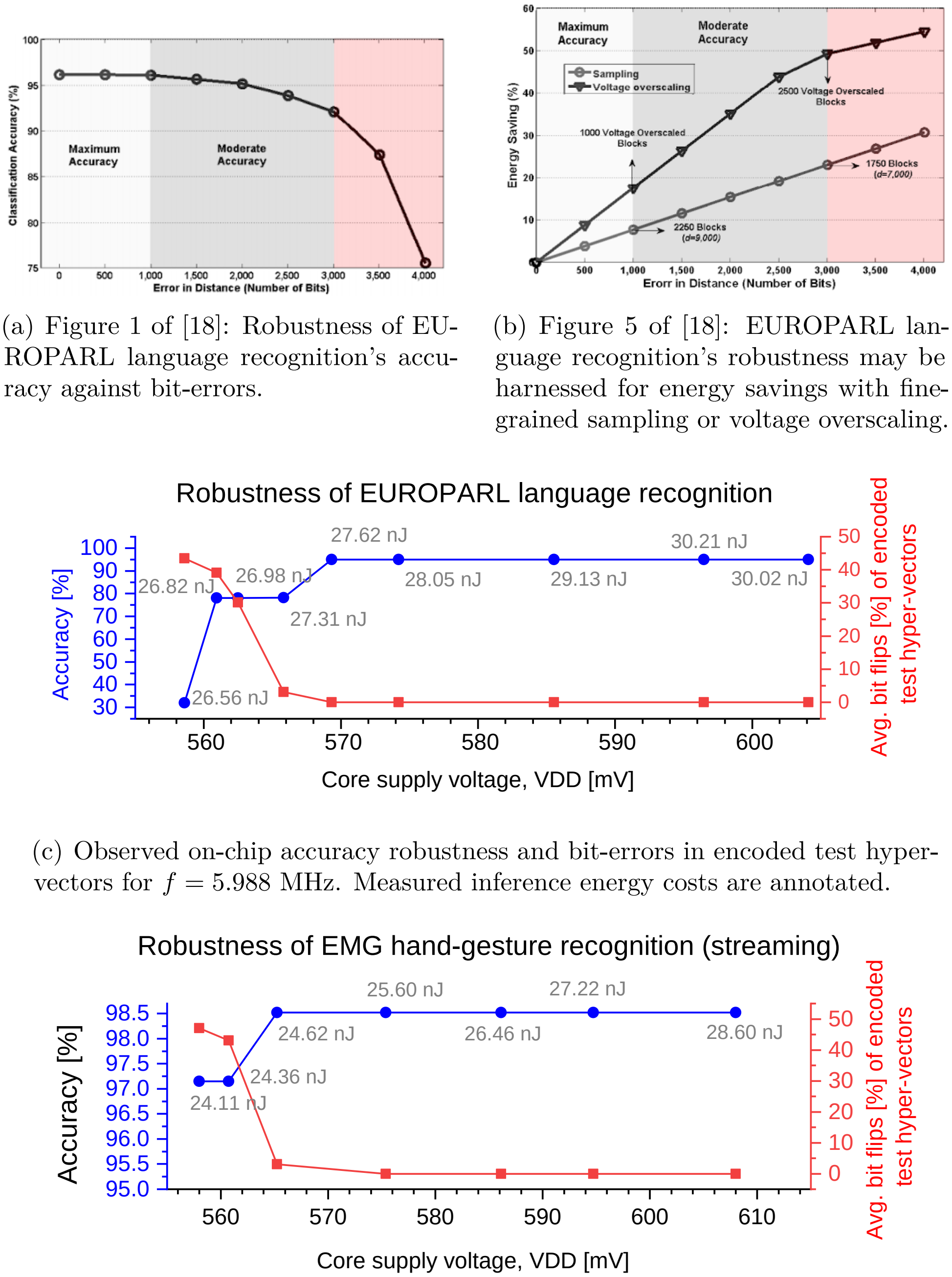
HDC literature reports that robustness of classification accuracy with respect to representation errors is remarkably high. Specifically, [18] simulated a resistive Associative Memory structure with fine-grained control of the logic gates' voltage supply and reported splendid robustness of EUROPARL language recognition accuracy as a function of the fraction of bits corrupted (the *combined* fraction of the test hyper-vector and all language hyper-vectors) during associative search. Their simulations revealed a *gradual decline* in classification accuracy with increasing bit-error rate during associative search (see figure 6-20(a)). Their fine-grained Associative Memory allowed simulations where voltage supply could be reduced beyond the nominal limits for only a few logic gates at a time, resulting in a substantial reduction in inference energy costs (see figure 6-20(b)). Similar conclusions on HDC robustness have been drawn from simulations of device variability [18, 21].

Since the smallest (error-free) inference energy was observed for $f = 5.988$ MHz and VDD ≈ 600 mV for both applications, VDD was reduced from ≈ 0.6V to introduce bit-errors. The `READ_ENCODER` mode (used in section 6.2.5) was employed to read-out and compare the chip's encoded hyper-vectors with simulated hyper-vectors for 64 random tests, revealing the fraction of bit errors introduced. Figure 6-20 (c), (d) plots the observed error-rates and classification accuracy (averaged over 5000 repetitions for each VDD voltage) for EUROPARL language recognition and streaming EMG hand-gesture recognition respectively.

Measurements show that bit-error rate in the test's encoded hyper-vector is *extremely sensitive* to small reductions in VDD (about 10%/mV) . Nevertheless, **the accuracy remains high for bit-error $\leq 40\%$ for both applications.** VDD ≤ 557 mV resulted in an output timeout by the chip – indicating a fatal control failure.

Publication:	Tran. BioCAS 2015[176]	Tran. CAS-II 2017[177]	Tran. BioCAS 2019[178]	Nature Electronics 2021[39]	Tran. CAS-I 2021[174]	Nature Electronics 2020[179]	**This work**
Electrode frequency	1 KHz	1.6 KHz	1 KHz	1 KHz	1 KHz	2 KHz	1 KHz
Features extracted	envelope	mean abs. value	root mean square	mean abs. value	mean abs. value	continous items [87]	mean abs. value
Data window	3 ms	$200-250$ ms	60 ms	250 ms	100 ms	$25-125$ ms	100 ms [38], 250 ms [39]
Inference algorithm	`float32` SVM	neural networks	HDC	HDC	HDC	HDC	HDC
HDC dimension	- (not HDC)	- (not HDC)	10000 bits	1000 bits	2048 bits	10000 bits	2048 bits
Number of channels	8, differential	4, differential	8, differential	64, single-ended	64, single-ended	4, single-ended	64, single-ended
Number of subjects	4 healthy	4 healthy	10 healthy	5 healthy (offline), 2 healthy (online)	3 healthy	5 healthy	3 healthy for [38] $5+2$ healthy for [39]
Number of gestures	6 + rest	10	10 + rest	20 + rest	4 + rest	4 + rest	4 + rest for [38] 20 + rest for [39]
Classification accuracy for tests	89.20%	94%	85%	84.53% (offline), 92.87% (online)	95.2%	98.9%	96.64% for [38], 84.53%, 92.87% for [39]
Platform for classification	90nm ARM Cortex M4 μ-controller	90nm ARM Cortex M4 μ-controller	40nm PULP v2 8-core cluster μ-controller [180]	FPGA (Microsemi M2S060T)	ASIC with 22nm CMOS	ASIC with 90nm PCM memory, 65nm CMOS logic	ASIC with 28nm CMOS
Data-path width	32 bits	32 bits	32 bits	- (FPGA)	2048 bits	10000 bits	2048 bits
Measurement type	directly from μ-controller	directly from μ-controller	directly from μ-controller	directly from FPGA	post-layout simulations	partly from chip(1)	directly from chip
Core supply	3.3 V	3.3 V	0.8 V	1.2 V	0.6 V	1.2V logic, 0.1V PCM	0.6 V
Pred. throughput	≈ 1000 preds/s	≈ 5000 preds/s	27778 preds/s	38462 preds/s	147.5 preds/s	$\approx 3.15 \times 10^7$ preds/s	44029 preds/s
Pred. latency	1 ms	0.2 ms	36 us	26 us	6.78 ms	-	118.07 us
Energy/pred.	89.1 μJ	13.8 μJ	83.2 μJ	1.95 μJ	191 nJ	13.3 nJ	28.59 ± 0.24 nJ
Energy/prediction for each channel	11.14 μJ	3.45 μJ	10.4 μJ	30.47 nJ	2.98 nJ	3.32 nJ	446.7 ± 3.7 pJ

Table 6.4: Comparison of measured energy per inference for EMG hand-gesture recognition. The reported energy/classification are for VDD = 0.6 V and `CLK` frequency is 5.988 MHz when streaming in test inputs for EMG hand-gesture recognition. All data sets except [39] use single-degree-of-freedom gestures. (1)The Phase-Change Memory (PCM) measurements in [70], implementing the Associative Memory and 2-minterm approximation (for binding n-grams during encoding), were done directly on chip – rest of its data-path was simulated post-synthesis in 65nm CMOS.



(a) Figure 1 of [18]: Robustness of EUROPARL language recognition's accuracy against bit-errors.

(b) Figure 5 of [18]: EUROPARL language recognition's robustness may be harnessed for energy savings with fine-grained sampling or voltage overscaling.

(c) Observed on-chip accuracy robustness and bit-errors in encoded test hyper-vectors for $f = 5.988$ MHz. Measured inference energy costs are annotated.

(d) Observed on-chip accuracy robustness and bit-errors in encoded test hyper-vectors for $f = 5.988$ MHz. Measured inference energy costs are annotated.

Figure 6-20: Measured robustness of classification accuracy with voltage over-scaling. In [18], EUROPARL language recognition's accuracy simulations with varying bit flips in encoded and language hyper-vectors during associative search showed significant tolerance (a) and potential for energy savings (b). Measurements from chip for EUROPARL language recognition (c) and streaming EMG hand-gesture recognition (d) for VDD ≤ 0.6V and `CLK` frequency $f = 5.988$ MHz are shown.

Chapter 7

Conclusions

The principal contributions of this dissertation may be summarized as follows:

1. **The *Generic* HDC architecture.** Inferring from the uniquely high-dimensional data-path width and symbolic nature of HDC operations, a *reasonable restriction* of arbitrarily-programmable architectures for efficient hyper-dimensional computing was specified. This body of arguments, described in chapters 3 and 4, form the foundational core of this study.

2. **A probability-inspired numerical normalization for integer HDC.** After identifying the core bottleneck in feasibility of multi-bit HDC data-paths, chapter 5 derives a method of numerical normalization of integer hyper-vectors stored in the Associative Memory. The proposed method is proven to be effective in *universally* limiting the bit-precision of the hyper-vectors' elements when their empirical distribution is reasonably Gaussian. This normalization could be utilized in an efficient integer processor for hyper-dimensional computing.

3. **A pioneering HDC processor was manufactured, tested and measured.** Chapter 6 describes the 2048-dimensional binary HDC chip fabricated in a 28nm technology node, which is a pioneering contribution in the following manner:

   (a) **This is the first chip that contains a HDC processor *in its entirety*.** All hyper-vectors are computed in full width. No off-chip memory is required for its functioning and no part of the HDC algorithm is computed outside the chip.

   In contrast, [136] and [21] do not support full-width hyper-vectors, and [70] simulates a portion of its data-path.

   (b) **This is the first HDC chip that can be programmed to handle a multitude of algorithms with a great diversity in data-rates and data channels.** All *generic* HDC algorithms complying with the uni-directional hyper-vector flow described in section 3.3.4; using ≤ 1024 random item hyper-vectors and ≤ 32 Associative Memory prototype hyper-vectors; and requiring at most 2 encoding stages, with ≤ 2 and ≤ 7 HLU layers in each respective stage, can be computed using this chip. In particular, this chip can perform all 7 supervised classification tasks listed in table 4.1 of section 4.3.1. In contrast, [21] and [136] can perform only language recognition.

Furthermore, *all supported encoding algorithms are computed exactly.* This is unlike [70] where only a 2-minterm approximation of the n-gram encoding algorithm can be implemented.

(c) **Measurements establish this chip to be the most energy-efficient for both tested applications.** In chapter 6, tables 6.3 and 6.4 present comparisons for both benchmarks tested on chip (see table 6.1). As per the author's knowledge, this chip is also the most efficient classifier as validated by measurements – for both language recognition of transliterated texts and EMG hand-gesture recognition – using *any classification algorithm* other than HDC.

The results reported in [70, 179] are a close second for both applications. However, they contain measurements for only the PCM memory in the data path – the rest was simulated after synthesis. Nevertheless, their estimates indicate a potential to surpass this chip in energy efficiency. It is yet be validated with on-chip measurements – preferably after adapting to the real-time data-rates of the applications (it currently has needlessly high prediction throughput).

(d) **Measurements establish HDC robustness for tested applications.** While [21] simulates the effect of RRAM variability (measured from chip) on the classification accuracy, [136] provided the first measured confirmation of robustness to stuck-at faults seen in chip's outputs.

Measurements on this chip (see figure 6-20) provides further evidence of HDC robustness for language recognition. Importantly, it furnishes *first-ever evidence for real-time robustness of EMG hand-gesture recognition.*

(e) **This is the first HDC processor designed and fabricated using *only* conventional digital logic and memory.** While [21, 136, 70] describe ASICs using emerging memory technologies such as PCM and RRAMs, or novel logic devices such as CNFETs, no known ASIC exists that was designed and manufactured using *only* conventional n-MOSFETs and p-MOSFETs.

Such an ASIC is necessary to **establish a competitive baseline** against which HDC systems designed and manufactured with advanced technology can be compared. This chip provides such a baseline; it would facilitate in isolating the novel technology's contribution to system's overall efficiency.

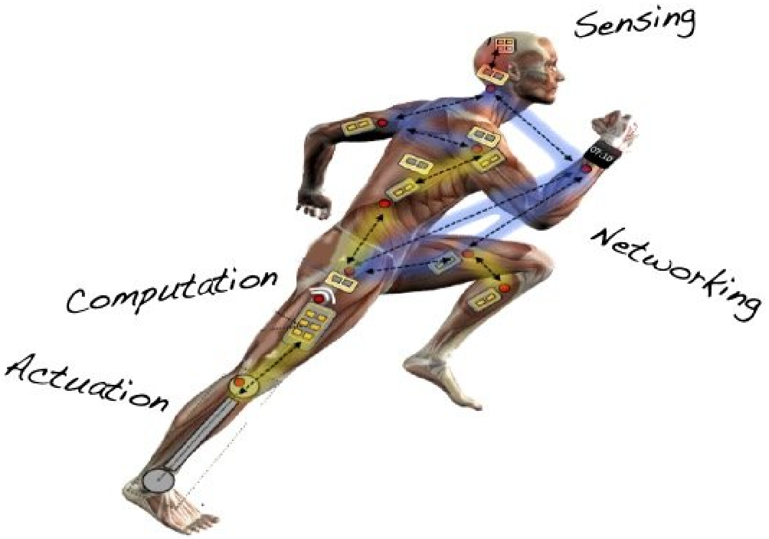


Figure 7-1: HDC is suitable paradigm for human-centric computing.
A network of intelligent sensors, actuators and efficient processors could form an on-body computing fabric in the future. An ultra-low-power processor core capable of functioning with energy harvested from the ambience is the key enabling technology.

First expressed in the abstract [181], an inter-connected network of intelligent sensors, actuators and *highly efficient* processors (illustrated in figure 7-1) could form a computing fabric on a user's person. Successful applications could be monitoring health and emotional state, recognize and assist in postures and daily activities and administering telemedicine. HDC was identified in [181] due to its potential for ultra-low-power processing at extremely energy-constrained environments.

The chip measurements presented in this dissertation confirm the effectiveness of HDC. By reliably achieving ≈ 30 nJ per gesture classification, it provides evidence that the technology may have matured to the point where a system like that is feasible. Such a project could begin by build on the setup described in [39] to create an on-body activity and health monitoring system. Electrode arrays could be stitched into the user's clothes, securely transmitting real-time data sensed from their entire body to a central hub containing an HDC processor chip, memory and battery. The processor could be used to train and classify incoming channel values and recognize the user's physical stress levels, activity states; or combine them with other types of body-sensing data. The classification labels may then be utilized to provide services to the user – such as estimate daily calories expended, suggest physical exercises and massages, monitor long-term health, and so on.

There are several other fruitful research directions where this work may be improved upon and extended in the author's opinion.

- The chip's Item and Associative memories are rudimentary. Sections 4.2.1 and 4.2.2 list a few advanced implementations for these memories respectively.
- An asynchronous or stochastic implementation of HDC processor is an interesting research direction. A comparison with such implementations with this work could reveal interesting design patterns for greater energy efficiency.
- Non-binary HDC implementations would be the natural next step of this chip. In particular, a design with the proposed normalization on-chip would be a great resource to analyze the efficiency and robustness estimated in section 5.4.
- The theoretical analysis of the proposed normalization in chapter 3 can be extended as well. A future work could generalize the result to other applications with prototype hyper-vectors obeying other distributions such as the truncated Normal, Beta and Pareto distributions and other members of Sub-Gaussian, Sub-Gamma and Exponential families.
- The Generic HDC architecture may be adapted and specialized for ultra-high efficiency in target applications. For instance, the EMG pre-processing logic could be augmented to a specialized HDC data-path for ultra-efficient gesture or activity recognition using EMG signals collected from a subjects body in real time.

  A similar design combining the pre-processor and the classifier for hand-gestures using visual frames captured by a camera in real time is described in [182]. However, they are expensive to train, are vulnerable to visual interference such as flashes and background artifacts, and reduce in effectiveness when the subject is far away, in vigorous kinetic motion or is occluded.
- Finally, a system of HDC processors as a part of a larger network of diverse computation cores, or hybrid designs for HDC and neural networks (such as [183]) are great project ideas.

Finally, this work required a sustained and elaborate design effort for the fabricated HDC processor chip. During this period, important lessons were learned that could greatly improve the productivity of designers and the quality of the overall design experience. The most prominent lessons are listed below:

1. When constructing an ASIC design, **include all necessary test modes and capabilities in the design**. This encourages the designers to develop a testing strategy *far earlier* in the design cycle, which in turn improves the productivity while writing RTL source code.

   Furthermore, any estimates and analysis of the design will always include the testing overhead – a *necessary overhead* for all hardware to function properly after manufacturing. Analysis without testing overhead is very likely to produce an (unreasonably) optimistic result.
2. **Ensure thorough testing of the design's functionality and performance-enhancement features.** Proper testing is necessary for designs intended to be manufactured and measured as the entire process is very expensive in time, designer-months, financial and computing resources. To organize this effort, it helps to sequentially number each feature that is intended in the design, beginning with the lowest-level feature. For composite features requiring multiple simple features, list all constituents explicitly.

   Similarly, it helps to number and administer tests in an order that progressively establishes correctness of the chip (with high confidence) – beginning with the lowest-level features. When designing a collection of tests for a complex or composite feature, ensure that the collection is *necessary and sufficient* to argue the feature's correctness.
3. **Complete a pass through all stage of silicon design as early as possible.** This helps new and inexperienced designers to gain familiarity with all the stages in silicon design early, and helps tremendously in identifying bottlenecks related to compute resources, EDA software versions and licenses, and physical design libraries.
4. **Strongly prefer any possible reuse of previous designs and cells.** Given the intensity of effort required, it is far more beneficial to reuse previously compiled and tested net-lists and silicon designs wherever possible. Reuse may be desirable over a new design even if a degradation in final efficiency is anticipated as it greatly increases the odds of successful completion.
5. **Plan the peripheral testing strategy *before* manufacturing the design.** The peripheral testing strategy includes designing all required printed-circuit boards, listing all required packages, PCB components and test equipments; and any special signal considerations when testing and measuring the chip. This greatly reduces the chance of errors in the testing setup *before* the irreversible and expensive step of committing a design for manufacturing.

Bibliography

[1] C. A. Mack, "Fifty years of moore's law," *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, pp. 202–207, May 2011.

[2] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[3] S. Salahuddin, K. Ni, and S. Datta, "The era of hyper-scaling in electronics," *Nature Electronics*, vol. 1, no. 8, p. 442, 2018.

[4] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, pp. 10–16, Nov 2005.

[5] T. N. Theis and H.-S. P. Wong, "The end of moore's law: A new beginning for information technology," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 41–50, 2017.

[6] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: from big data to big impact," *MIS quarterly*, pp. 1165–1188, 2012.

[7] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[8] Y.-B. Kim, "Challenges for nanoscale mosfets and emerging nanoelectronics," *Transactions on Electrical and Electronic Materials*, vol. 11, no. 3, pp. 93–105, 2010.

[9] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, pp. 253–266, Feb 2010.

[10] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pp. 365–376, June 2011.

[11] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[12] R. Sarpeshkar, "Analog versus digital: extrapolating from electronics to neurobiology," *Neural computation*, vol. 10, no. 7, pp. 1601–1638, 1998.

[13] R. Echeveste, L. Aitchison, G. Hennequin, and M. Lengyel, "Cortical-like dynamics in recurrent circuits optimized for sampling-based probabilistic inference," *Nature neuroscience*, vol. 23, no. 9, pp. 1138–1149, 2020.

[14] 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.

[15] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals," *Proceedings of the IEEE*, vol. 107, pp. 123–143, Jan 2019.

[16] 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.

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

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

[19] 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*, ISLPED '16, (New York, NY, USA), pp. 64–69, ACM, 2016.

[20] T. F. Wu, H. Li, P. C. Huang, A. Rahimi, J. M. Rabaey, H. S. P. Wong, M. M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pp. 492–494, Feb 2018.

[21] H. Li, T. F. Wu, A. Rahimi, K. S. Li, M. Rusch, C. H. Lin, J. L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, W. C. Chiu, M. C. Chen, T. T. Wu, J. M. Shieh, W. K. Yeh, J. M. Rabaey, S. Mitra, and H. S. P. Wong, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *2016 IEEE International Electron Devices Meeting (IEDM)*, pp. 16.1.1–16.1.4, Dec 2016.

[22] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), pp. 108:1–108:6, ACM, 2018.

[23] M. Radovanović, A. Nanopoulos, and M. Ivanović, "On the existence of obstinate results in vector space models," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, (New York, NY, USA), p. 186–193, Association for Computing Machinery, 2010.

[24] V. Pestov, "Is the k-nn classifier in high dimensions affected by the curse of dimensionality?," *Computers & Mathematics with Applications*, vol. 65, no. 10, pp. 1427–1437, 2013. Grasping Complexity.

[25] I. Assent, "Clustering high dimensional data," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 4, pp. 340–350, 2012.

[26] G. Geenens, "Curse of dimensionality and related issues in nonparametric functional regression," *Statistics Surveys*, vol. 5, pp. 30–43, 2011.

[27] T. A. Plate, "Holographic reduced representations," *IEEE Transactions on Neural networks*, vol. 6, no. 3, pp. 623–641, 1995.

[28] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[29] D. A. Rachkovskij, "Representation and processing of structures with binary sparse distributed codes," *IEEE transactions on Knowledge and Data Engineering*, vol. 13, no. 2, pp. 261–276, 2001.

[30] P. Blouw, E. Solodkin, P. Thagard, and C. Eliasmith, "Concepts as semantic pointers: A framework and computational model," *Cognitive science*, vol. 40, no. 5, pp. 1128–1162, 2016.

[31] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," *arXiv preprint cs/0412059*, 2004.

[32] S. Datta, R. A. G. Antonio, A. R. S. Ison, and J. M. Rabaey, "A programmable hyper-dimensional processor architecture for human-centric iot," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 439–452, 2019.

[33] A. DasGupta, *Normal Approximations and the Central Limit Theorem*, pp. 213–242. New York, NY: Springer New York, 2010.

[34] M. Okamoto, "Some inequalities relating to the partial sum of binomial probabilities," *Annals of the institute of Statistical Mathematics*, vol. 10, no. 1, pp. 29–35, 1959.

[35] P. Kanerva, "Some properties of the space $\{0, 1\}^n$," in *Sparse distributed memory*, ch. 1, pp. 18–22, MIT press, 1988.

[36] M. Ledoux, *The concentration of measure phenomenon.* No. 89, American Mathematical Soc., 2001.

[37] 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.

[38] A. Moin, A. Zhou, A. Rahimi, S. Benatti, A. Menon, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt, L. Benini, A. C. Arias, and J. M. Rabaey, "An emg gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2018.

[39] A. Moin, A. Zhou, A. Rahimi, A. Menon, S. Benatti, G. Alexandrov, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, *et al.*, "A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition," *Nature Electronics*, vol. 4, no. 1, pp. 54–63, 2021.

[40] A. Menon, D. Sun, M. Aristio, H. Liew, K. Lee, and J. M. Rabaey, "A highly energy-efficient hyperdimensional computing processor for wearable multi-modal classification," in *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4, 2021.

[41] Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 115–120, 2020.

[42] S. Gupta, M. Imani, B. Khaleghi, V. Kumar, and T. Rosing, "Rapid: A reram processing in-memory architecture for dna sequence alignment," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, 2019.

[43] I. Barclay, C. Simpkin, G. Bent, T. La Porta, D. Millar, A. Preece, I. Taylor, and D. Verma, "Trustable service discovery for highly dynamic decentralized workflows," *Future Generation Computer Systems*, 2022.

[44] A. Burrello, K. Schindler, L. Benini, and A. Rahimi, "Hyperdimensional computing with local binary patterns: One-shot learning of seizure onset and identification of ictogenic brain regions using short-time ieeg recordings," *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 2, pp. 601–613, 2020.

[45] A. Burrello, L. Cavigelli, K. Schindler, L. Benini, and A. Rahimi, "Laelaps: An energy-efficient seizure detection algorithm from long-term human ieeg recordings without false alarms," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 752–757, 2019.

[46] D. Kleyko, C. Bybee, C. J. Kymn, B. A. Olshausen, A. Khosrowshahi, D. E. Nikonov, F. T. Sommer, and E. P. Frady, "Integer factorization with compositional distributed representations," *arXiv preprint arXiv:2203.00920*, 2022.

[47] Y. Yao, W. Liu, G. Zhang, and W. Hu, “Radar-based human activity recognition using hyperdimensional computing,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 70, no. 3, pp. 1605–1619, 2022.

[48] P. R. Genssler and H. Amrouch, “Brain-inspired computing for circuit reliability characterization,” *IEEE Transactions on Computers*, pp. 1–1, 2022.

[49] D. Ma, S. Zhang, and X. Jiao, “Hdcoin: A proof-of-useful-work based blockchain for hyperdimensional computing,” *arXiv preprint arXiv:2202.02964*, 2022.

[50] W. He, Y. Ye, T. Pan, Q. Meng, and Y. Li, “Emotion recognition from ecg signals contaminated by motion artifacts,” in *2021 International Conference on Intelligent Technology and Embedded Systems (ICITES)*, pp. 125–130, 2021.

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

[52] K. Schlegel, F. Mirus, P. Neubert, and P. Protzel, “Multivariate time series analysis for driving style classification using neural networks and hyperdimensional computing,” in *2021 IEEE Intelligent Vehicles Symposium (IV)*, pp. 602–609, 2021.

[53] 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)*, pp. 384–389, 2021.

[54] M. Hersche, S. Lippuner, M. Korb, L. Benini, and A. Rahimi, “Near-channel classifier: symbiotic communication and classification in high-dimensional space,” *Brain Informatics*, vol. 8, no. 1, pp. 1–15, 2021.

[55] F. Cumbo, E. Cappelli, and E. Weitschek, “A brain-inspired hyperdimensional computing approach for classifying massive dna methylation data of cancer,” *Algorithms*, vol. 13, no. 9, p. 233, 2020.

[56] M. Hersche, E. M. Rella, A. Di Mauro, L. Benini, and A. Rahimi, “Integrating event-based dynamic vision sensors with sparse hyperdimensional computing: A low-power accelerator with online learning capability,” in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED ’20, (New York, NY, USA), p. 169–174, Association for Computing Machinery, 2020.

[57] Y. Ma, M. Hildebrandt, V. Tresp, and S. Baier, “Holistic representations for memorization and inference.,” in *UAI*, pp. 403–413, 2018.

[58] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, “Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception,” *Science Robotics*, vol. 4, no. 30, p. eaaw6736, 2019.

[59] B. Cheung, A. Terekhov, Y. Chen, P. Agrawal, and B. A. Olshausen, “Superposition of many models into one,” *CoRR*, vol. abs/1902.05522, 2019.

[60] M. Hersche, M. Zeqiri, L. Benini, A. Sebastian, and A. Rahimi, “A neuro-vector-symbolic architecture for solving raven’s progressive matrices,” *arXiv preprint arXiv:2203.04571*, 2022.

[61] M. Hersche, G. Karunaratne, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, “Constrained few-shot class-incremental learning,” 2022.

[62] J. Yang, Y. Sheng, S. Zhang, R. Wang, K. Foreman, M. Paige, X. Jiao, W. Jiang, and L. Yang, “Automated architecture search for brain-inspired hyperdimensional computing,” *arXiv preprint arXiv:2202.05827*, 2022.

[63] P. Kanerva, “What we mean when we say" what’s the dollar of mexico?": Prototypes and mapping in concept space.,” in *AAAI fall symposium: quantum informatics for cognitive, social, and semantic processes*, pp. 2–6, 2010.

[64] 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,” *arXiv preprint arXiv:2111.06077*, 2021.

[65] A. Joshi, J. T. Halseth, and P. Kanerva, “Language geometry using random indexing,” in *International Symposium on Quantum Interaction*, pp. 265–274, Springer, 2016.

[66] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[67] T. Vatanen, J. J. Väyrynen, and S. Virpioja, “Language identification of short text segments with n-gram models.,” in *LREC*, 2010.

[68] J. F. D. Silva and G. P. Lopes, “Identification of document language is not yet a completely solved problem,” in *2006 International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA’06)*, pp. 212–212, Nov 2006.

[69] S. F. Chen and J. Goodman, “An empirical study of smoothing techniques for language modeling,” *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.

[70] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, “In-memory hyperdimensional computing,” *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

[71] G. Karunaratne, A. Rahimi, M. L. Gallo, G. Cherubini, and A. Sebastian, “Real-time language recognition using hyperdimensional computing on phase-change memory array,” in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–1, 2021.

[72] G. Karunaratne, M. Schmuck, M. Le Gallo, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, “Robust high-dimensional memory-augmented neural networks,” *Nature communications*, vol. 12, no. 1, pp. 1–12, 2021.

[73] X. Yin, F. Müller, Q. Huang, C. Li, M. Imani, Z. Yang, J. Cai, M. Lederer, R. Olivo, N. Laleni, *et al.*, “An ultra-compact single fefet binary and multi-bit associative search engine,” *arXiv preprint arXiv:2203.07948*, 2022.

[74] M. Eggimann, A. Rahimi, and L. Benini, “A 5 uw standard cell memory-based configurable hyperdimensional computing accelerator for always-on smart sensing,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4116–4128, 2021.

[75] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, “F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing,” in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA ’19, (New York, NY, USA), p. 53–62, Association for Computing Machinery, 2019.

[76] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, “A binary learning framework for hyperdimensional computing,” in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 126–131, 2019.

[77] P. Poduval, M. Issa, F. Imani, C. Zhuo, X. Yin, H. Najafi, and M. Imani, “Robust in-memory computing with hyperdimensional stochastic representation,” in *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 1–6, 2021.

[78] A. Hernández-Cano, C. Zhuo, X. Yin, and M. Imani, *Real-Time and Robust Hyperdimensional Classification*, p. 397–402. New York, NY, USA: Association for Computing Machinery, 2021.

[79] Y. Kim, M. Imani, and T. S. Rosing, “Efficient human activity recognition using hyperdimensional computing,” in *Proceedings of the 8th International Conference on the Internet of Things*, IOT ’18, (New York, NY, USA), pp. 38:1–38:6, ACM, 2018.

[80] S. Benatti, F. Montagna, V. Kartsch, A. Rahimi, D. Rossi, and L. Benini, “Online learning and classification of emg-based gestures on a parallel ultra-low power platform using hyperdimensional computing,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 3, pp. 516–528, 2019.

[81] M. Hersche, E. M. Rella, A. Di Mauro, L. Benini, and A. Rahimi, “Integrating event-based dynamic vision sensors with sparse hyperdimensional computing: A low-power accelerator with online learning capability,” in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED ’20, (New York, NY, USA), p. 169–174, Association for Computing Machinery, 2020.

[82] F. Montagna, A. Rahimi, S. Benatti, D. Rossi, and L. Benini, “Pulp-hd: Accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform,” in *Proceedings of the 55th Annual Design Automation Conference*, DAC ’18, (New York, NY, USA), pp. 111:1–111:6, ACM, 2018.

[83] P. Poduval, H. Alimohamadi, A. Zakeri, F. Imani, M. H. Najafi, T. Givargis, and M. Imani, “Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning,” *Frontiers in Neuroscience*, vol. 16, 2022.

[84] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, “Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals,” *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, 2019.

[85] 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)*, pp. 221–234, 2021.

[86] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, “Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals,” *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, 2019.

[87] 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 *Rebooting Computing (ICRC), IEEE International Conference on*, pp. 1–8, IEEE, 2016.

[88] A. Moin, A. Zhou, S. Benatti, A. Rahimi, L. Benini, and J. M. Rabaey, “Analysis of contraction effort level in emg-based gesture recognition using hyperdimensional computing,” in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4, 2019.

[89] “Prior releases of european parliament proceedings parallel corpus.” `https://www.statmt.org/europarl/archives.html`. Accessed: 30 January 2022.

[90] “European parliament proceedings parallel corpus 1996-2011.” `https://www.statmt.org/europarl/index.html`. Accessed: 30 January 2022.

[91] J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR ’12, (Washington, DC, USA), pp. 3642–3649, IEEE Computer Society, 2012.

[92] D. Ciregan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649, 2012.

[93] D. Franklin, "Nvidia jetson tx2 delivers twice the intelligence to the edge." `https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/`, 2017.

[94] Texas Instruments, *INA226 High-Side or Low-Side Measurement, Bi-Directional Current and Power Monitor with I2C Compatible Interface*, 2011.

[95] A. O. Troy Hanson, "Ut-hash: A hash table for c structures." `https://troydhanson.github.io/uthash/`, 2017.

[96] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[97] "Kann." `https://github.com/attractivechaos/kann`, 2018.

[98] "perf: Linux profiling with performance counters." `https://perf.wiki.kernel.org/index.php/Main_Page`.

[99] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *ACM Sigplan notices*, vol. 42, no. 6, pp. 89–100, 2007.

[100] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[101] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen, "ThunderSVM: A fast SVM library on GPUs and CPUs," *Journal of Machine Learning Research*, vol. 19, pp. 1–5, 2018.

[102] M. D. Hill and V. J. Reddi, "Accelerator-level parallelism," *Commun. ACM*, vol. 64, p. 36–38, nov 2021.

[103] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, (New York, NY, USA), p. 53–62, Association for Computing Machinery, 2019.

[104] D. Rossi, I. Loi, A. Pullini, C. Müller, A. Burg, F. Conti, L. Benini, and P. Flatresse, "A self-aware architecture for pvt compensation and power nap in near threshold processors," *IEEE Design Test*, vol. 34, pp. 46–53, Dec 2017.

[105] A. Coates and A. Y. Ng, “The importance of encoding versus training with sparse coding and vector quantization,” in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 921–928, 2011.

[106] G. E. Batista, R. C. Prati, and M. C. Monard, “A study of the behavior of several methods for balancing machine learning training data,” *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[107] C. E. Shannon, “The synthesis of two-terminal switching circuits,” *The Bell System Technical Journal*, vol. 28, no. 1, pp. 59–98, 1949.

[108] E. P. Frady, S. J. Kent, B. A. Olshausen, and F. T. Sommer, “Resonator Networks, 1: An Efficient Solution for Factoring High-Dimensional, Distributed Representations of Data Structures,” *Neural Computation*, vol. 32, pp. 2311–2331, 12 2020.

[109] H. V. Jagadish, S. K. Rao, and T. Kailath, “Array architectures for iterative algorithms,” *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1304–1321, 1987.

[110] U. Eckhardt and R. Merker, “Hierarchical algorithm partitioning at system level for an improved utilization of memory structures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 1, pp. 14–24, 1999.

[111] S. V. Rajopadhye, “Synthesizing systolic arrays with control signals from recurrence equations,” *Distributed Computing*, vol. 3, no. 2, pp. 88–105, 1989.

[112] S. Borkar, R. Cohn, G. Cox, S. Gleason, and T. Gross, “Warp: An integrated solution of high-speed parallel computing,” in *Proceedings of the 1988 ACM/IEEE Conference on Supercomputing*, Supercomputing ’88, (Los Alamitos, CA, USA), pp. 330–339, IEEE Computer Society Press, 1988.

[113] K. K. Parhi, C.-Y. Wang, and A. P. Brown, “Synthesis of control circuits in folded pipelined dsp architectures,” *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, 1992.

[114] P. Cappello, “A processor-time-minimal systolic array for cubical mesh algorithms,” *IEEE transactions on parallel and distributed systems*, vol. 3, no. 1, pp. 4–13, 1992.

[115] T. Komarek and P. Pirsch, “Array architectures for block matching algorithms,” *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp. 1301–1308, 1989.

[116] H. Jagadish and T. Kailath, “A family of new efficient arrays for matrix multiplication,” *IEEE Transactions on computers*, vol. 38, no. 1, pp. 149–155, 1989.

[117] M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, and T. Rosing, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 190–198, 2019.

[118] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446, 2019.

[119] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing designs against scan-based side-channel attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 325–336, 2007.

[120] E.-J. Chang, A. Rahimi, L. Benini, and A.-Y. A. Wu, "Hyperdimensional computing-based multimodality emotion recognition with physiological signals," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 137–141, 2019.

[121] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer, "Computing on functions using randomized vector representations," *ArXivorg*.

[122] "Ent: A pseudorandom number sequence test program." `https://www.fourmilab.ch/random/`. Accessed: 06 March 2022.

[123] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, S. Leigh, M. Levenson, M. Vangel, N. Heckert, and D. Banks, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," 2010-09-16 2010.

[124] "Dieharder: A random number test suite." `https://webhome.phy.duke.edu/~rgb/General/dieharder.php`. Version: 3.31.1, Accessed: 06 March 2022.

[125] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, p. 3–30, jan 1998.

[126] K. Rikitake, "Tinymt pseudo random number generator for erlang," in *Proceedings of the Eleventh ACM SIGPLAN Workshop on Erlang Workshop*, Erlang '12, (New York, NY, USA), p. 67–72, Association for Computing Machinery, 2012.

[127] M. Saito and M. Matsumoto, "Simd-oriented fast mersenne twister: a 128-bit pseudorandom number generator," in *Monte Carlo and Quasi-Monte Carlo Methods 2006* (A. Keller, S. Heinrich, and H. Niederreiter, eds.), (Berlin, Heidelberg), pp. 607–622, Springer Berlin Heidelberg, 2008.

[128] M. Saito and M. Matsumoto, "Variants of mersenne twister suitable for graphic processors," *ACM Trans. Math. Softw.*, vol. 39, feb 2013.

[129] M. Tomassini, M. Sipper, and M. Perrenoud, “On the generation of high-quality random numbers by two-dimensional cellular automata,” *IEEE Transactions on Computers*, vol. 49, no. 10, pp. 1146–1151, 2000.

[130] J. C. Cerda, C. D. Martinez, J. M. Comer, and D. H. K. Hoe, “An efficient fpga random number generator using lfsrs and cellular automata,” in *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 912–915, 2012.

[131] S. Wolfram, “A new kind of science,” 2002.

[132] W. Meier and O. Staffelbach, “Analysis of pseudo random sequences generated by cellular automata,” in *Advances in Cryptology — EUROCRYPT ’91* (D. W. Davies, ed.), (Berlin, Heidelberg), pp. 186–199, Springer Berlin Heidelberg, 1991.

[133] M. Sipper and M. Tomassini, “Generating parallel random number generators by cellular programming,” *International Journal of Modern Physics C*, vol. 07, 10 1996.

[134] D. Kleyko, E. P. Frady, and F. T. Sommer, “Cellular automata can reduce memory requirements of collective-state computing,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2021.

[135] A. Menon, D. Sun, M. Aristio, H. Liew, K. Lee, and J. M. Rabaey, “A highly energy-efficient hyperdimensional computing processor for wearable multi-modal classification,” in *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4, 2021.

[136] T. F. Wu, H. Li, P.-C. Huang, A. Rahimi, G. Hills, B. Hodson, W. Hwang, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, “Hyperdimensional computing exploiting carbon nanotube fets, resistive ram, and their monolithic 3d integration,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 11, pp. 3183–3196, 2018.

[137] M. Dichtl and J. D. Golić, “High-speed true random number generation with logic gates only,” in *Cryptographic Hardware and Embedded Systems - CHES 2007* (P. Paillier and I. Verbauwhede, eds.), (Berlin, Heidelberg), pp. 45–62, Springer Berlin Heidelberg, 2007.

[138] C. Tokunaga, D. Blaauw, and T. Mudge, “True random number generator with a metastability-based quality control,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 78–85, 2008.

[139] M. Stipčević and Ç. K. Koç, *True Random Number Generators*, pp. 275–315. Cham: Springer International Publishing, 2014.

[140] C. Camara, H. Martín, P. Peris-Lopez, and M. Aldalaien, “Design and analysis of a true random number generator based on gsr signals for body sensor networks,” *Sensors*, vol. 19, no. 9, 2019.

[141] M. Rusch, "The design of an analog associative memory circuit for applications in high-dimensional computing," Master's thesis, EECS Department, University of California, Berkeley, May 2018.

[142] T. Yu, Y. Zhang, Z. Zhang, and C. De Sa, "Understanding hyperdimensional computing for parallel single-pass learning," *arXiv preprint arXiv:2202.04805*, 2022.

[143] S. J. Kent, E. P. Frady, F. T. Sommer, and B. A. Olshausen, "Resonator Networks, 2: Factorization Performance and Capacity Compared to Optimization-Based Methods," *Neural Computation*, vol. 32, pp. 2332–2388, 12 2020.

[144] N. Chamidah and I. Wasito, "Fetal state classification from cardiotocography based on feature extraction using hybrid k-means and support vector machine," *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pp. 37–41, 2015.

[145] A. M. Bagirov, J. Ugon, D. Webb, and B. Karasözen, "Classification through incremental max–min separability," *Pattern Analysis and Applications*, vol. 14, pp. 165–174, May 2011.

[146] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*, pp. 216–223, Springer, 2012.

[147] 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)*, pp. 1–8, Nov 2017.

[148] Y. Kim, M. Imani, and T. Rosing, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 25–32, Nov 2017.

[149] J. M. Rabaey, "The human intranet–where swarms and humans meet," *IEEE Pervasive Computing*, vol. 14, no. 1, pp. 78–83, 2015.

[150] E. P. Frady, D. Kleyko, and F. T. Sommer, "A theory of sequence indexing and working memory in recurrent neural networks," *Neural Computation*, vol. 30, no. 6, pp. 1449–1513, 2018.

[151] M. Hersche, S. Lippuner, M. Korb, L. Benini, and A. Rahimi, "Near-channel classifier: symbiotic communication and classification in high-dimensional space," *Brain Informatics*, vol. 8, no. 1, pp. 1–15, 2021.

[152] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on electronic Computers*, no. 1, pp. 14–17, 1964.

[153] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, "Designing arithmetic building blocks," in *Digital integrated circuits: a design perspective*, ch. 11, pp. 578–589, Pearson Education, Inc., 2nd ed., 2003.

[154] S. Zhang, R. Wang, J. J. Zhang, A. Rahimi, and X. Jiao, "Assessing robustness of hyperdimensional computing against errors in associative memory : (invited paper)," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 211–217, 2021.

[155] A. Kazemi, M. M. Sharifi, Z. Zou, M. Niemier, X. S. Hu, and M. Imani, "Mimhd: Accurate and efficient hyperdimensional inference using multi-bit in-memory computing," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, 2021.

[156] S. Bosch, A. S. de la Cerda, M. Imani, T. S. Rosing, and G. D. Micheli, "Qubithd: A stochastic acceleration method for HD computing-based machine learning," *CoRR*, vol. abs/1911.12446, 2019.

[157] M. Imani, S. Bosch, S. Datta, S. Ramakrishna, S. Salamat, J. M. Rabaey, and T. Rosing, "Quanthd: A quantization framework for hyperdimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2268–2278, 2020.

[158] M. Hersche, L. Benini, and A. Rahimi, "Binarization methods for motor-imagery brain–computer interface classification," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 567–577, 2020.

[159] S. Boucheron, G. Lugosi, and P. Massart, *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

[160] M. J. Wainwright, *High-dimensional statistics: A non-asymptotic viewpoint*, vol. 48. Cambridge University Press, 2019.

[161] B. Laurent and P. Massart, "Adaptive estimation of a quadratic functional by model selection," *Annals of Statistics*, pp. 1302–1338, 2000.

[162] M. Beeler, R. W. Gosper, and R. Schroeppel, "Hakmem," 1972.

[163] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[164] P. Diaconis and S. Zabell, "Closed form summation for classical distributions: variations on a theme of de moivre," *Statistical Science*, pp. 284–302, 1991.

[165] Z. Zou, Y. Kim, F. Imani, H. Alimohamadi, R. Cammarota, and M. Imani, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[166] J. Morris, R. Fernando, Y. Hao, M. Imani, B. Aksanli, and T. Rosing, "Locality-based encoder and model quantization for efficient hyper-dimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 897–907, 2022.

[167] A. G. Anderson and C. P. Berg, "The high-dimensional geometry of binary neural networks," *arXiv preprint arXiv:1705.07199*, 2017.

[168] C. Maxfield, "An introduction to different rounding algorithms," *Programmable Logic Design Line*, pp. 1–15, 2006.

[169] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE transactions on computers*, vol. 100, no. 8, pp. 786–793, 1973.

[170] "Opal kelly xem7310." `https://opalkelly.com/products/xem7310/`. Accessed: 02 May 2022.

[171] "Ina229 85-v, 20-bit, ultra-precise power/energy/charge monitor with spi interface." `https://www.ti.com/lit/ds/symlink/ina229.pdf`. Accessed: 02 May 2022.

[172] "1.5a ultra-ldo with programmable sequencing." `https://www.ti.com/lit/ds/symlink/tps74301.pdf`. Accessed: 02 May 2022.

[173] "The opal kelly frontpanel software design kit." `https://opalkelly.com/products/frontpanel/`. Accessed: 02 May 2022.

[174] M. Eggimann, A. Rahimi, and L. Benini, "A 5 uw standard cell memory-based configurable hyperdimensional computing accelerator for always-on smart sensing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4116–4128, 2021.

[175] U. Quasthoff, M. Richter, and C. Biemann, "Corpus portal for search in monolingual corpora.,"

[176] S. Benatti, F. Casamassima, B. Milosevic, E. Farella, P. Schönle, S. Fateh, T. Burger, Q. Huang, and L. Benini, "A versatile embedded platform for emg acquisition and gesture recognition," *IEEE transactions on biomedical circuits and systems*, vol. 9, no. 5, pp. 620–630, 2015.

[177] X. Liu, J. Sacks, M. Zhang, A. G. Richardson, T. H. Lucas, and J. Van der Spiegel, "The virtual trackpad: An electromyography-based, wireless, real-time, low-power, embedded hand-gesture-recognition system using an event-driven artificial neural network," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 11, pp. 1257–1261, 2017.

[178] S. Benatti, F. Montagna, V. Kartsch, A. Rahimi, D. Rossi, and L. Benini, “Online learning and classification of emg-based gestures on a parallel ultra-low power platform using hyperdimensional computing,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 3, pp. 516–528, 2019.

[179] G. Karunaratne, M. Le Gallo, M. Hersche, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, “Energy efficient in-memory hyperdimensional encoding for spatio-temporal signal processing,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1725–1729, 2021.

[180] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gürkaynak, A. Teman, J. Constantin, A. Burg, I. Miro-Panades, E. Beignè, F. Clermidy, P. Flatresse, and L. Benini, “Energy-efficient near-threshold parallel computing: The pulpv2 cluster,” *IEEE Micro*, vol. 37, no. 5, pp. 20–31, 2017.

[181] J. Rabaey, A. Rahimi, S. Datta, M. Rusch, P. Kanerva, and B. Olshausen, “Human-centric computing — the case for a hyper-dimensional approach,” in *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, pp. 29–29, 2017.

[182] Y. Lu, V. L. Le, and T. T.-H. Kim, “9.7a 184uw real-time hand-gesture recognition system with hybrid tiny classifiers for smart wearable devices,” in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, pp. 156–158, 2021.

[183] G. Karunaratne, M. Schmuck, M. Le Gallo, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, “Robust high-dimensional memory-augmented neural networks,” *Nature communications*, vol. 12, no. 1, pp. 1–12, 2021.