UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Hardware-Algorithm Co-design for Efficient and Privacy-Preserved Edge Computing

Permalink

https://escholarship.org/uc/item/4rz7q6zx

Author

Khaleghi, Behnam

Publication Date

2022

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

Hardware-Algorithm Co-design for Efficient and Privacy-Preserved Edge Computing

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Behnam Khaleghi

Committee in charge:

Professor Tajana Simunic Rosing, Chair
Professor Chung-Kuan Cheng
Professor Ryan Kastner
Professor Farinaz Koushanfar
Professor Jishen Zhao

2022

Copyright
Behnam Khaleghi, 2022
All rights reserved.

The dissertation of Behnam Khaleghi is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2022

DEDICATION

To my wife, Fatemeh, for her unconditional love, care, and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to first express my gratitude to my advisor, Prof. Tajana Rosing, for her support, guidance, and appreciation during my Ph.D. She went above and beyond her responsibilities as an academic advisor and was understanding of all aspects of my life. Her kind and considerate attitude always impacted me throughout my Ph.D. I would like to also thank my committee members, Prof. CK Cheng, Prof. Ryan Kastner, Prof. Farinaz Koushanfar, and Prof. Jishen Zhao for their feedback and discussions related to this Ph.D. work. I would like to thank all my friends and colleagues in SEELab for their active collaboration. A very special thanks to Mohsen Imani and Amin Kalantar for their continued help and all the good memories. I would like to extend my thanks to all our collaborators from academia and industry who helped us during the last few years, especially Prof. Niema Moshiri, Ramesh Chauhan my supervisor when I was an intern at Qualcomm and his later supports, and Carlos Diaz and his team from TSMC.

My research was made possible by funding from the National Science Foundation (NSF) grants 1730158, 1911095, 2028040, 2003279, 2100237, CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, SRC Global Research Collaboration (GRC), DARPA HyDREA project, and Taiwan Semiconductor Manufacturing Company (TSMC).

Above all, I owe so much to my wife and my family. My wife, Fatemeh, has been unconditionally supportive and caring and really made countless sacrifices to help me get to this point. I wouldn't be able to pass overcome the difficulties without of her kindness, support, and patience.

The material in this dissertation is, in part, based on the following published or under review papers. The Abstract and Introduction constrain material from all works listed below. Chapter 2, in part, is a reprint of the material as it appears in "FPGA Energy Efficiency by Leveraging Thermal Margin" by Behnam Khaleghi, Sahand Salamat, Mohsen Imani, and Tajana Rosing, which appears in IEEE International Conference on Computer Design (ICCD), 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the material as it appears in "SHEARer: Highly-Efficient Hyperdimensional Computing by Software-Hardware Enabled Multifold Approximation" by Behnam Khaleghi, Sahand Salamat, Anthony Thomas, Fatemeh Asgarinejad, Yeseong Kim, and Tajana Rosing which appears in Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in "GENERIC: Highly Efficient Learning Engine on Edge using Hyperdimensional Computing" by Behnam Khaleghi, Jaeyoung Kang, Hanyang Xu, Justin Morris, and Tajana Rosing, which appears in Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC), 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in full, is currently being prepared for submission for publication of the material "Private Learning and Inference with Hyperdimensional Computing" by Behnam Khaleghi, Jaeyoung Kang, Xiaofan Yu, and Tajana Rosing, to be submitted to IEEE Internet of Things Journal. The dissertation author was the primary investigator and author of this paper.

VITA

2013	B. S. in Computer Engineering, Sharif University of Technology, Tehran, Iran
2016	M. S. in Computer Engineering (Computer Architecture), Sharif University of Technology, Tehran, Iran
2022	Ph. D. in Computer Science (Computer Engineering), University of California San Diego, US

PUBLICATIONS

B. Khaleghi, J. Kang, X. Yu, T. Rosing, "Private Learning and Inference with Hyperdimensional Computing", *IEEE Internet of Things Journal (IOTJ)*, 2022 (under submission).

B. Khaleghi, T. Zhang, C. Martino, G. Armstrong, A. Akel, K. Curewitz, J. Eno, S. Eilert, R. Knight, N. Moshiri, T. Rosing, "SALIENT: Ultra-Fast FPGA-based Short Read Alignment", *IEEE International Conference on Field-Programmable Technology (ICFPT)*, 2022.

A. Thomas, B. Khaleghi, G. K. Jha, N. Himayat, R. Iyer, N. Jain, T. Rosing, "Streaming Encoding Algorithms for Scalable Hyperdimensional Computing", *arXiv preprint*, 2022.

U. Mallappa, P. Gangwar, B. Khaleghi, H. Yang, T. Rosing, "TermiNETor: Early Convolution Termination for Efficient Deep Neural Networks", *IEEE International Conference on Computer Design (ICCD)*, 2022.

B. Khaleghi, T. Zhang, N. Shao, A. Akel, K. Curewitz, J. Eno, S. Eilert, N. Moshiri, T. Rosing, "FAST: FPGA-based Acceleration of Genomic Sequence Trimming", *IEEE Biomedical Circuits and Systems (BIOCAS)*, 2022.

J. Kang, B. Khaleghi, Y. Kim, T. Rosing, "OpenHD: A GPU-Powered Framework for Hyperdimensional Computing", *IEEE Transactions on Computers (TC)*, 2022.

J. Morris, Y. Hao, S. Gupta, B. Khaleghi, B. Aksanli, T. Rosing. "Stochastic-HD: Leveraging Stochastic Computing on the Hyper-Dimensional Computing Pipeline", *Frontiers in Neuroscience*, 2022.

A. Dutta, S. Gupta, B. Khaleghi, R. Chandrasekaran, W. Xu, T. Rosing. "HDnn-PIM: Efficient in Memory Design of Hyperdimensional Computing with Feature Extraction", *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2022.

J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, T. Rosing. "HyDREA: Utilizing Hyperdimensional Computing for a More Robust and Efficient Machine Learning System", *ACM Transactions on Embedded Computing Systems (TETC)*, 2022.

G. Armstrong, C. Martino, J. Morris, B. Khaleghi, et al. "Swapping Metagenomics Preprocessing Pipeline Components Offers Speed and Sensitivity Increases", *mSystems*, 2022.

B. Khaleghi*, U. Mallappa*, D. Yaldiz, H. Yang, M. Shah, J. Kang, T. Rosing "PatterNet: Explore and Exploit Filter Patterns for Efficient Deep Neural Networks", *59th ACM/IEEE Design Automation Conference (DAC)*, 2022.

B. Khaleghi, J. Kang, H. Xu, J. Morris, T. Rosing "GENERIC: Highly Efficient Learning Engine on Edge using Hyperdimensional Computing", *59th ACM/IEEE Design Automation Conference (DAC)*, 2022.

S. Gupta, B. Khaleghi, S. Salamat, J. Morris, R. Ramkumar, J. Yu, A. Tiwari, J. Kang, M. Imani, B. Aksanli, T. Rosing "Store-n-Learn: Classification and Clustering with Hyperdimensional Computing across Flash Hierarchy", *ACM Transactions on Embedded Computing Systems (TETC)*, 2022.

J. Morris, H. Lui, K. Stewart, B. Khaleghi, A. Thomas, T. Marback, B. Aksanli, E. Neftci, T. Rosing, "HyperSpike: HyperDimensional Computing for More Efficient and Robust Spiking Neural Networks", *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.

J. Kang, B. Khaleghi, Y. Kim, T. Rosing, "XCelHD: An Efficient GPU-Powered Hyperdimensional Computing with Parallelized Training", *27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022.

R. Fielding-Miller, S. Karthikeyan, T. Gaines, et al., "Wastewater and surface monitoring to detect COVID-19 in elementary school settings: The Safer at School Early Alert project", *Medrxiv*, 2021.

A. Paul, G. Hota, B. Khaleghi, Y. Xu, T. Rosing, G. Cauwenberghs, "Attention State Classification with In-Ear EEG", *IEEE Biomedical Circuits and Systems (BIOCAS)*, 2021.

Y. Hao, S. Gupta, J. Morris, B. Khaleghi, B. Aksanli, T. Rosing, "Stochastic-HD: Leveraging Stochastic Computing on Hyper-Dimensional Computing", *IEEE International Conference on Computer Design (ICCD)*, 2021.

S. Salamat, A. H. Aboutalebi, B. Khaleghi, J. H. Lee, Y. S. Ki, T. Rosing, "NASCENT: Near-Storage Acceleration of Database Sort on SmartSSD", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

B. Khaleghi, H. Xu, J. Morris, T. Rosing, "tiny-HD: Ultra-Efficient Hyperdimensional Computing Engine for IoT Applications", *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, T. Rosing, "HyDREA: Towards More Robust and Efficient Machine Learning Systems with Hyperdimensional Computing", *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

R. Garcia, F. Asgarinejad, B. Khaleghi, T. Rosing, M. Imani, "TruLook: A Framework for Configurable GPU Approximation", *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

S. Salamat, S. Shubhi, B. Khaleghi, T. Rosing, "Residue-Net: Multiplication-free Neural Network by In-situ, No-loss Migration to Residue Number Systems", *27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021.

B. Khaleghi, S. Salamat, T. Rosing, "Revisiting FPGA Routing under Varying Operating Conditions", *International Conference on Field-Programmable Technology (ICFPT)*, 2020.

B. Khaleghi, S. Salamat, A. Thomas, F. Asgarinejad, Y. Kim, T. Rosing, "SHEARer: Highly-Efficient Hyperdimensional Computing by Software-Hardware Enabled Multifold AppRoximation", *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2020.

B. Khaleghi, M. Imani, T. Rosing, "Prive-HD: Privacy-Preserved Hyperdimensional Computing", *ACM/IEEE Design Automation Conference (DAC)*, 2020.

B. Khaleghi, S. Salamat, M. Imani, T. Rosing, "FPGA Energy Efficiency by Leveraging Thermal Margin", *International Conference on Computer Design (ICCD)*, 2019.

S. Salamat, B. Khaleghi, M. Imani, T. Rosing, "Workload-Aware Opportunistic Energy Efficiency in Multi-FPGA Platforms", *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

S. Gupta, M. Imani, B. Khaleghi, V. Kumar, T. Rosing, "RAPID: A ReRAM Processing in Memory Architecture for DNA Sequence Alignment", *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019.

M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, T. Rosing, "SparseHD: Algorithm-Hardware Co-Optimization for Efficient High-Dimensional Computing", *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

J. Sim, M. Kim, Y. Kim, S. Gupta, B. Khaleghi, T. Rosing, "MAPIM: Mat Parallelism for High Performance Processing in Non-volatile Memory Architecture", *International Symposium on Quality Electronic Design (ISQED)*, 2019.

S. Salamat, M. Imani, B. Khaleghi, T. Rosing, "F5-HD: Fast Flexible FPGA-based Framework for Refreshing Hyperdimensional Computing", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

B. Khaleghi, T. Rosing, "Thermal-Aware Design and Flow for FPGA Performance Improvement", *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

ABSTRACT OF THE DISSERTATION

Hardware-Algorithm Co-design for Efficient and Privacy-Preserved Edge Computing

by

Behnam Khaleghi

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California San Diego, 2022

Professor Tajana Simunic Rosing, Chair

The rapidly growing number of edge devices continuously generating data with real-time response constraints coupled with the bandwidth, latency, and reliability issues of centralized cloud computing have made computing near the edge indispensable. As a result, using Field-Programmable Gate Arrays (FPGAs) at the edge, due to their unique capabilities that meet the requirements of both high-performance applications and the Internet of Things (IoT) domain, is becoming prevalent. However, designs deployed on these devices suffer from efficiency gap versus custom implementations mainly due to the overhead associated with the FPGAs reconfigurability. This problem is more pronounced in the edge domain, where most devices are battery-powered. In the first part of this dissertation, we identify and overcome the challenges

behind the power reduction of FPGA-based applications and propose techniques to lower their energy consumption. Our approach exploits the pessimistic timing margin of the designs to tune the voltage and improves the energy consumption by 66%.

An increasing number of edge applications rely on machine learning (ML) algorithms to generate useful insights from data. While modern machine learning techniques – in particular deep neural networks (DNNs) – can produce state-of-the-art results, they often entail substantial memory and compute requirements that may exceed the power and resources available on lightweight error-prone edge devices. Hyperdimensional Computing (HDC) is an emerging lightweight and robust learning paradigm suited for the edge domain that copes with the memory and compute overhead of conventional ML algorithms. The next part of the dissertation proposes efficient FPGA-based and custom hardware implementations of HDC to enable intelligence on devices with limited resources, strict energy constraints, and in noisy environments. The proposed HDC algorithms and accelerators reduce the energy consumption by more than three orders of magnitude compared to other ML solutions, with a comparable or better accuracy.

The last part of the dissertation seeks to resolve the privacy concerns of HDC that stem from its reversible algorithm and pose challenges for HDC-based learning and inference. We propose hardware- and communication-efficient techniques that improve the 'inference' privacy of HDC by reducing the information of the transferred data while consuming less energy than the non-private baseline. We then show that HDC 'learning' can meet tight privacy budgets with negligible accuracy degradation. We also propose a hybrid CNN and HDC model for differentially-private training over image data, which achieves comparable or better accuracy than the state-of-the-art CNN-only methods with more than three orders of magnitude faster training.

Chapter 1

Introduction

The growing number of computing devices and systems in the information age creates an exploding amount of data every moment. There are already over 13 billion connected devices of different sizes and computation power, from sensory edge devices to many-core servers, producing or consuming around 3000 terabyte of data every second [4, 5]. Processing such an unprecedented volume of data has exacerbated the energy and performance issues of current processing systems caused by the failure of Dennard scaling [6] and the slowdown of Moore's law [7]. Notably, the growth of the edge devices, ranging from resource and energy-constrained lightweight remote sensors carrying out simple data filtering to battery-powered self-driving cars performing complex machine learning (ML) algorithms, have aggravated the bandwidth, latency, and reliability challenges of offloading data to the cloud and made more efficient and robust computing at the edge indispensable [8, 9].

A promising approach to address the energy and performance crisis is architectural specialization [10], i.e., designing a custom accelerator tailored for particular application(s). While an effective solution for ML and AI market, developing customized hardware is costly, time-consuming, and hence not plausible for applications with limited use cases or continually

evolving/updating ones. Accordingly, using Field-Programmable Gate Arrays (FPGAs), which offer programmability while being more energy efficient than CPUs and GPUs [11, 12] is getting more popular not only on cloud [13, 14, 15] but also at the edge [16, 17, 18]. However, the reconfigurability of FPGAs causes non-negligible performance and energy gaps with the custom ASIC counterparts [19, 20]. In addition, nowadays, more and more edge applications rely on ML algorithms to generate useful insights from data [21, 22, 23]. While modern ML techniques – in particular deep neural networks (DNNs) – can produce state-of-the-art results, they often entail substantial memory and compute requirements that exceed edge devices' resource and energy limitations even if implemented on custom architectures [24, 25, 26, 27].

That being said, there is a pressing need to unfold more aggressive energy reduction techniques on both the hardware and algorithm side while considering the reliability and privacy concerns. In this dissertation, we seek orthogonal hardware and algorithm strategies to achieve this goal. Given the FPGAs gaining traction in both data center and edge applications, in the first part of this dissertation, we identify and overcome the challenges behind the power reduction of FPGA-based applications and propose techniques to lower their energy consumption. The second part of the dissertation focuses on alternative computing paradigms to address the resource and energy demands of nowadays ubiquitous algorithms, i.e., machine learning. To this end, we introduce hyperdimensional computing (HDC) as a lightweight and robust learning alternative and propose efficient FPGA-based and custom ASIC implementations of HDC to enable intelligence on devices with limited resources, strict energy constraints, and in noisy environments. We also investigate and address the privacy challenges of HD computing.

Field-Programmable Gate Arrays (FPGAs): The key characteristics of FPGAs, such as customizability based on the applications features, programmability to adopt new attributes, and low latency by creating deep processing pipeline, have paved the way for these devices to be deployed on data centers to either provide software as a service such as low-latency DNN accelerator [14] and search engine [13] or used internally such as offloading host networking to

hardware [15]. That being said, FPGAs are even more attractive at the edge. Compared to the other commercial off-the-shelf alternatives such as microcontrollers, FPGAs offer versatile interfaces to connect peripherals such as cameras and other types of sensors. The connectivity of FPGAs also allows them to act as a support processing unit for sensory devices in the edge gateway [18]. The unrivaled computing power offered by FPGAs, together with their reconfigurability that allows in-field upgrade, make them more advantageous for a manifold of edge applications including but not limited to image and signal processing, communication within the Internet of things (IoT) stack, smart energy controller system, and real-time heavyweight video analysis [18]. Therefore, more and more applications in the IoT era can be implemented on FPGAs, with the need for utmost energy efficiency [16, 17, 18]. However, the reconfigurability of FPGAs, while appealing, imposes energy overhead. This, together with slowed shrinking of supply voltage [28], have pushed the high-end family of these devices to a point they consume power comparable to processors [29]. All in all, more aggressive power reduction approaches for FPGAs have become indispensable.

Hyperdimensional Computing (HDC): Hyperdimensional Computing (HDC) is a novel brain-inspired learning paradigm based on the observation that brains perform cognitive tasks by mapping sensory inputs to high-dimensional neural representation [30, 31, 32, 33, 34, 35]. It enables the brain to carry out simple, low-power, error-resilient, and parallelizable operations, all in hyperspace. Such characteristics of HDC make it appealing for a wide variety of applications, particularly the edge domain, with devices with tight resource and energy constraints working in noisy environments. HDC uses a deterministic encoding to map the raw input data to points in hyperspace, represented by vectors with thousands of dimensions. For random vectors $\vec{H}_1$ and $\vec{H}_2$, the sum vector $\vec{S} = \vec{H}_1 + \vec{H}_2$ is more similar to $\vec{H}_1$ and $\vec{H}_2$ than any other random vector. This concept is leveraged to represent sets in hyperspace. Vectors of the same category are bundled by vector addition to create a class vector. To classify a query, the input is encoded in the same fashion and compared with the class vectors to find the most similar label.

Despite the simplicity of the HDC algorithm, its bit-level massively parallel operations do not accord well with general-purpose processors such as CPUs and GPUs due to, e.g., memory latency and data movement of large vectors and the fact that these devices are over-provisioned for majorly binary operations of HDC. FPGA-based implementation can provide a great extent of flexibility, high degree of parallelism, and bit-level granularity of operations that significantly improves the effective allocation of resources and hence the performance. Nevertheless, while appealing for a large body of edge use cases, FPGAs suffer from large form-factor and leakage power, as well as power overhead associated with its reconfiguration capability. These drawbacks hinder the deployment of FPGAs in extremely energy/area-constrained use cases such as wearable devices and ultra-low-power sensory devices and call for custom HDC circuits. However, previous attempts on custom HDC accelerators support a limited number of applications, achieve low accuracy, and mainly do not support training [36, 37, 38, 39, 40].

Additionally, the simple encoding techniques that HDC benefits from are reversible to the original data. That is, the original data can be reconstructed from the encoding vectors with a good estimate. It poses serious privacy concerns to data. First, in certain applications of HDC, such as federated learning or cloud-hosted inference [41, 42], offloading the encoded data jeopardizes privacy since an untrustworthy server or communication link can reconstruct the original data. Second, as the HDC model is a superposition of vectors, small changes in the training data can reveal the statistics of particular encoded vectors from which the raw data can be retrieved. It is more problematic than DNNs, wherein the model is realized by backpropagation and non-linear operations that somewhat obfuscate the training details [43].

In the rest of the introduction, we give an overview of the contributions of this dissertation to tackle the aforementioned challenges.

1.1 Efficiency by Aggressive Energy Reduction of FPGA-based Designs

FPGA devices are continuously evolving to meet the high computation and performance demand for emerging applications. As a result, cutting-edge FPGAs are not as energy efficient as conventionally presumed, and therefore, aggressive power-saving techniques have become imperative. The clock rate of an FPGA-mapped design is set based on worst-case conditions to ensure reliable operation under all circumstances. This usually leaves a considerable timing margin that can be exploited to reduce power consumption by scaling voltage without lowering clock frequency. There are hurdles for such opportunistic voltage scaling in FPGAs because (a) critical paths change with designs, making timing evaluation difficult as voltage changes, (b) each FPGA resource has a particular power-delay trade-off with voltage, (c) data corruption of configuration cells and memory blocks further hampers voltage scaling.

Chapter 2 details the proposed techniques to reduce the power and energy consumption of FPGA-based designs. To this end, we propose incorporating thermal-aware voltage scaling in the FPGA design flow. We first obtain the temperature-delay-voltage correlation of FPGA resources within the supported temperature range. Then, we statically (i.e., offline) estimate the thermal distribution of applications to obtain the available timing headroom, for which voltages of different power rails can be efficiently determined based on the characterized library. For further effectiveness, we also propose online (i.e., dynamic) voltage adaptation based on the response of thermal sensors. The proposed methods consider the voltage-temperature feedback loop and the separate power rails of specific resource types to yield maximum power efficiency. Thereafter, we leverage the aforementioned proposed flow to explore the *energy* consumption, whereby we trade-off the performance and power consumption to achieve the minimum energy point. This is desirable for a majority of edge and IoT applications for which total energy consumption is of the utmost concern. Experimental results over a set of industrial benchmarks indicate up to

36% power reduction with the same performance and 66% total energy saving when energy is the optimization target.

Our approaches mentioned above are deterministic, as they guarantee timing closure. Nonetheless, many use cases, such as image processing and machine learning applications, can tolerate a certain level of computation errors, which allows voltage *over-scaling*. However, it needs an examination of applications under these non-ideal conditions to get a glimpse of produced error in the output. Accordingly, we propose a primary FPGA simulation framework to be able to evaluate an FPGA-mapped design under voltage-scaling, i.e., when the delay of resources varies. Using the proposed framework, we map demonstrative machine learning applications into FPGA fabric and examine the impact of voltage *over-scaling* on extra power gain versus accuracy drop. This yields an additional $\sim$15% power reduction compared to our original (non-invasive) voltage scaling.

1.2 Efficiency by a New Learning Algorithm for Edge Applications

1.2.1 Exact and Approximate FPGA Implementation of HDC Inference

The primary appeal of HD computing lies in its amenability to implementation in modern hardware accelerators. Because the HD representations are simply long Boolean vectors, they can be processed extremely efficiently in highly parallel platforms. Particularly, FPGA-based implementations can provide a high degree of parallelism and bit-level customization of operations that significantly improves the effective resource utilization and hence the performance. The principal challenge of HD computing – and the focus of this chapter – lies in designing good encoding schemes that (1) represent the data in a format suitable for learning and (2) are efficient to implement in hardware. In general, the encoding phase is the most expensive stage in the HD

learning pipeline. Existing encoding methods require generating vectors in full integer-precision and then *ex-post* quantizing to $\{\pm 1\}$. While this accelerates the associative search phase, it does not address encoding, which is the primary source of inefficiency.

In Chapter 3, we propose novel techniques to compute the encodings in an approximate manner that saves a substantial amount of resources with an insignificant impact on accuracy. Of independent interest is our novel FPGA implementation that achieves striking performance through massive parallelism with low power consumption. Approximate encodings entail models to be trained in a similar approximate fashion. Thus we also develop a software emulation to enable users to train desired HD models. Our software framework enables users to explore the trade-off between the degree of approximation, accuracy, and resource utilization (hence power consumption) by generating a pre-compiled library that correlates approximation schemes and FPGA resource utilization and power consumption. We show that our procedure leads to performance improvement of $15.7\times$ and energy savings of up to $301\times$ compared to state-of-the-art encoding methods implemented on GPU.

1.2.2 Highly-Efficient ASIC Design of HDC Learning and Inference

Unleashing the potential of HDC is contingent on the underlying hardware. Conventional processing platforms such as CPUs and GPUs are incapable of taking full advantage of the highly-parallel bit-level operations of HDC. FPGA platforms, while appealing for HDC, cannot be utilized in the domains that require small form-factor and extremely low power and energy consumption such as tiny wearable devices. While this makes custom HDC engines crucial, existing HDC encoding techniques do not cover a broad range of applications to make a custom HDC design capable of running varied applications without the need for expensive FPGA-like reconfigurability plausible.

In Chapter 4, we propose `GENERIC` (highly efficient learning engine on edge using hyperdimensional computing) for highly efficient and accurate trainable classification and cluster-

ing. Our primary goal is to make GENERIC compact and low-power to meet year-long battery-powered operation, yet fast enough during training and burst inference, e.g., when it serves as an IoT gateway. To this end, we first propose a novel HDC encoding that yields high accuracy in various benchmarks. Such a generic encoding is fundamental to developing a custom yet flexible circuit and avoiding otherwise expensive configurable (FPGA-like) platforms. Then we perform a detailed comparison of HDC and various ML techniques on conventional devices and point out the failure of these devices in unleashing HDC advantages. To our knowledge, it is the first detailed comparison of the accuracy and efficiency of different HDC and ML techniques. Using the proposed encoding, we develop the GENERIC flexible architecture that implements accurate HDC-based trainable classification and clustering. GENERIC benefits from extreme energy reduction techniques such as application-opportunistic power gating, on-demand dimension reduction, and error-resilient voltage over-scaling.

We implemented GENERIC using 14 nm technology. The design occupies only 0.30 mm^2 area and consumes 90 μW static and 3.94 μW/MHz dynamic power. Comparison of GENERIC with the state-of-the-art HDC implementations reveals GENERIC improves the classification accuracy by at least 3.5% over existing HDC techniques and 6.5% over conventional ML techniques (e.g., support-vector machines, random forest, and multilayer perceptron). Compared to the most efficient baseline ML algorithm (random forest), GENERIC improves the inference energy consumption by 530×, and training energy by 1590× with 8.2% higher accuracy. Such a tiny form-factor and power draw of our design enables uninterrupted use of HDC in a very wide-range of IoT applications including wearables and remote sensors that otherwise would require network access (and associated peripherals) and battery replacement.

1.2.3 Preserving the Privacy of HDC-based Learning and Inference

One of the unique advantages that HDC offers is facilitating federated learning, where distributed edge devices can simply offload the encoded vectors and a central aggregator creates

the unified model by just adding up the vectors and returning the aggregate model. Federated learning by HDC significantly reduces the transmission costs, requires less computation power, and can cope with unreliable network due to the error tolerance of HDC [41]. In addition, in certain cases that the memory of a very tiny sensory device might not fit the HDC model, vectors can be offloaded to an adjacent device, gateway or cloud for inference. Since HDC uses simple encoding techniques, as we shown in Chapter 5, these encoded vectors can be decoded to the original data with a good estimate, which raises privacy concerns to data since an untrustworthy server or communication link can reconstruct the original data. Second, as the HDC model (classes) is a superposition of vectors, small changes in the training data can reveal the statistics of particular encoded vectors from which the raw data can be retrieved. It is more problematic than DNNs, wherein the model is realized by backpropagation and non-linear operations that can obfuscate the training details.

In Chapter 5, we investigate the inference and training privacy of HDC. We show that although HDC operations are reversible, we can leverage its noise resiliency for private training and inference. Specifically, we make the following novel contributions. First, for different HDC encoding techniques, we show that *encoding parameters* can be extracted by using adversary inputs and observing the encoded vector. Then, using these extracted (or already known) parameters, we introduce techniques to decode the *original data* from the encoded vectors for various encodings. The decoded data have relatively low mean squared error (MSE) versus the original data. To tackle this reversibility challenge, we propose hardware- and communication-efficient techniques to enhance the inference privacy of HDC by reducing the information of the transferred data. To this end, we use hardware-friendly (regular) local sparsification of the encoded vectors that eliminates all but the maximum value of consecutive vector embeddings. Our approach, implemented on FPGA, reduces both encoding and communication energy (by transferring the index of the maximum element only). Thereafter, the chapter targets privacy-preserved training of HDC based on the concept of differential privacy. We propose private one-pass (suited for online

and federated learning) as well as iterative learning of HDC. We show that privacy-preserved HDC training, in particular one-pass learning, can meet tight privacy budgets with a similar accuracy to the non-private models. Finally, we propose privacy-preserved HDC models for image classification, namely Privé-HDnn (HD + neural network), which combines a pre-trained CNN feature extractor and a trainable HDC classifier for private training on the target dataset. By using a constant CNN trained on public dataset and private training of HDC only, we get around the sensitivity of CNNs to noise injection while extracting complex features for HDC to tackle its challenge on image data. Experimental results show Privé-HDnn can achieve a similar or better (by up to 5.8%) accuracy compared to CNN-only approaches, with 9.2–1231$\times$ faster training, while offering other advantages such as few-shot learning due to the capability of HDC in learning from fewer data.

Chapter 2

FPGA Energy Efficiency by Leveraging Thermal Margin

In the previous chapter, we explained that the programmability and performance balance of FPGAs have paved the way for these devices to a variety of heavy data center applications such as search engines, machine learning, and networking [13, 14, 15] to name a few. Other key characteristics of FPGAs, such as easy interfacing with peripherals and integration with soft and hard processors, have also made the lower-end families of devices popular in less heavy edge workloads [16, 17, 18]. Unfortunately, the resource overhead associated with the programmability of devices has caused an energy efficiency gap between the designs implemented on FPGAs versus the ASIC counterparts. In this chapter, we dig into some roots of this inefficiency and mitigation challenges and elaborate on our approach to reduce the energy consumption of FPGA devices, both data center class and lower-end devices. We propose a systematical approach to leverage the available thermal headroom of FPGA-mapped designs for power and energy improvement by tuning the voltage level. By comprehensively analyzing the timing and power consumption of FPGA building blocks under varying temperatures and voltages, we propose a thermal-aware

voltage scaling flow that effectively utilizes the thermal margin to reduce power consumption without degrading performance. We show that the proposed flow can be especially employed for energy optimization as well, whereby power consumption and delay are compromised to accomplish the tasks with minimum energy.

2.1 Introduction

The prevalence of computation-intensive workloads with high-performance requirements such as machine learning (ML) and data center applications [13, 14, 15, 44] accompanied with the advance of technology node have persuaded the FPGA vendors to integrate more resources with boosted clock rate in state-of-the-art FPGAs [45]. This, together with slowed shrinking of FPGAs supply voltage [28], have pushed these devices to a point they consume power comparable to CPUs [29]. Beside data center applications, there are prevailing energy-constrained applications at the edge, implemented in more low-end FPGAs, with the need for extreme energy efficiency [16, 17].

As FPGAs already employ a manifold of device-level optimization to throttle power consumption [46], more *aggressive* power reduction techniques are gaining traction. These techniques generally build upon the conservative timing margin (d_g) that is considered to compensate reliability threats in deep-nano technologies; while an FPGA-mapped design is able to deliver an *actual* clock period of $d_{V_{nom}}$ at nominal voltage V_{nom}, in practice, STA (static timing analysis) tools report an operating clock delay of $d_{V_{nom}} + d_g$ to make up for uncertainties such as voltage fluctuations, degradation, temperature, etc. [29]. The aforementioned aggressive techniques exploit this available timing headroom to reduce supply voltage of FPGAs down to V_{low} for which the *actual* clock period $d_{V_{low}}$ becomes equal or close to $d_{V_{nom}} + d_g$ (leaving no margin for guardbands). Thus, the device still delivers the original performance at a lower voltage. Note that *aggressive* voltage scaling techniques are different from conventional DVFS (dynamic voltage and frequency

scaling) that concurrently tunes the frequency and voltage based on per-task performance demand of applications.

Although lowering the supply voltage of processors and ASIC devices has been known to be an effective power saving technique [47, 48, 49], there are limited studies presenting voltage scaling in FPGAs. Voltage scaling (in particular, aggressive and performance-aware one) in FPGAs is challenging mainly because: **(a)** *Critical Path* (CP) in an FPGA is design-dependent, which makes timing probing difficult under voltage scaling, especially considering the impact of temperature. Therefore, in contrast to ASIC designs, a set of precalibrated stand-alone sensory circuits, e.g., ring oscillators and CP monitors [48] cannot accurately correlate the sensor frequency with all varying CPs. **(b)** FPGA architectures are heterogeneous, comprising soft-fabric (i.e., programmable logic and routing resources), DSP cores, memory blocks (BRAM), etc. These building blocks are tightly coupled, and each has a particular power/delay relation with supply voltage. Considering the separate voltage rails provided for certain components, i.e., V_{bram}, V_{core}, and V_{io} that can be regulated separately, finding efficient voltage points becomes design-dependent and challenging. In other words, in multi-supply devices, multiple voltage combinations can lead to the target $d_{V_{nom}} + d_g$ boundary, while only one tuple yields minimum power. This makes speculative voltage decrement no more efficient. **(c)** Scaling the voltage of FPGAs is also constrained by the data corruption of configuration and memory SRAM cells. In addition, as we will discuss in this chapter, reducing the voltage of configuration cells unexpectedly increases FPGA power consumption in certain cases, calling for cautious analysis.

We leverage the pessimistic thermal-induced timing slack to scale down FPGA operating voltage for power saving while tackling the aforementioned challenges as follows.

(1) We propose to incorporate thermal-aware voltage scaling in the FPGA design flow. We first obtain the temperature-delay-voltage correlation of FPGA resources within the supported temperature range. Then, we statically estimate the thermal distribution of applications to obtain the available timing headroom, for which voltages of different power rails can be efficiently

determined based on the characterized library. For further effectiveness, we also suggest online (i.e., dynamic) voltage adaptation based on the response of thermal sensors. The proposed methods consider the voltage-temperature feedback loop and the separate power rails of specific resource types to yield maximum power efficiency.

(2) We leverage the proposed flow of (1) to explore the *energy* consumption, whereby we trade-off the performance and power consumption to achieve the minimum energy point. This is desirable for a majority of edge and IoT applications for which total energy consumption is the utmost concern.

(3) Our approaches mentioned in (1) and (2) are deterministic, as they guarantee timing closure. Nonetheless, many use-cases such as image processing and machine learning applications can tolerate a certain level of computation errors, which gives an opportunity for voltage *over-scaling*. However, it needs an examination of applications under these non-ideal conditions to get a glimpse of produced error in the output. We propose a primary FPGA simulation framework to be able to evaluate an FPGA-mapped design under voltage-scaling, i.e., when the delay of resources vary. Using the proposed framework, we map demonstrative machine learning applications into FPGA fabric and examine the impact of voltage *over-scaling* on extra power gain versus accuracy drop.

2.2 Background and Related Work

2.2.1 FPGA Architecture

Figure 2.1 illustrates the architecture of conventional tile-based FPGAs. The architecture comprises tiles of logic clusters (a.k.a CLBs or slices) that bind together using the configurable switch boxes (SBs) and connection blocks (CBs) to implement larger functions. Each logic CLB consists of N (e.g., $N = 10$) K-input look-up tables (LUTs) each of which is capable to implement Boolean expressions up to K variables. SB multiplexers are located in the intersection of horizontal and vertical channels (wire tracks) to enable connectivity and bending of nets.

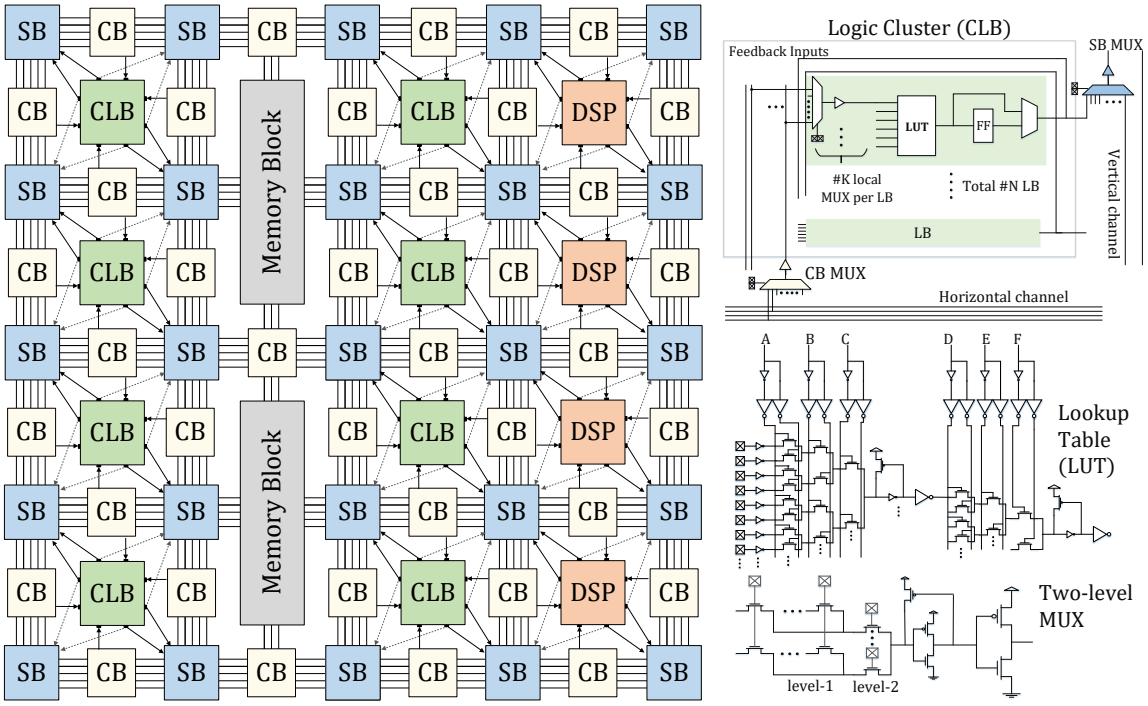


Figure 2.1: Tile-based FPGA architecture (left), and building blocks (right) [1].

These multiplexers are also responsible for connecting outputs of logic resources (LUTs, FFs, carry chain, etc.) to global routing, i.e., to horizontal and vertical channels. Analogously, CB multiplexers pass the selected global wires into the logic clusters. Specific FPGA columns are repetitively dedicated to Block RAMs and DSP cores. The majority of logic and routing resources have a multiplexer-like structure, mainly implemented as two-stage multiplexers that have been shown to provide optimal area-delay efficiency [50]. These resources often drive large loads, especially the global routing resources that have a high number of long fanout wires, so are augmented with large output buffers to improve performance.

2.2.2 Related Work

While many previous studies have attempted to reduce FPGA power dissipation by power-gating, conventional DVFS, configurable dual supply voltage, control of signal levels, architectural innovations, etc. [51], there are limited number of works that leverage the available

timing margin to reduce voltage without scarifying performance. Authors of [52] explore the timing headroom of FPGA-mapped designs by gradually reducing the voltage (keeping the frequency fixed) until observing the error. They detect the error by inserting a shadow register per each CP with a phase-shifted clock to detect the data mismatch caused by voltage reduction. As there are a huge number of CP and near-CP paths in a large synthesis-flattened design, the area and power overhead of this technique can be excessive. Also, measuring the slack in this technique depends on capturing the signal traversing the CP, while the CP may not be controllable at the runtime. Furthermore, the introduced capture registers cannot be used for paths that head to hard blocks such as BRAM. The study in [53] addresses the latter issue by replicating such paths and inserting an end-point proxy register (instead of the hard block) where an error-detector circuitry checks timing violations. However, the error-detector itself imposes delay, so a timing error raised in the original CP might propagate into the memory before being detected.

In [54], the authors propose a two-step self-calibrating voltage scaling scheme by exploiting the available timing slack of thermal margin. CPs of a design are extracted by using the STA tool and are then implemented on the FPGA fabric. Afterward, for different values of temperature and core voltage (T and V_{core}), the maximum frequency is obtained by gradually increasing it until error is observed by the implemented error-detection circuit. This approach does not consider the thermal distribution within the chip [55] where a CP may experience varied temperature in reality. This either results in timing violation by ignoring the parts of CP that resides in hot (hence, slower) tiles, or gives non-optimal results if a related thermal margin is considered. In addition, the STA tool reports the CPs according to worst-case condition while CPs might change at lower temperatures. Thus a larger number of near-CP paths need to evaluate which makes the entire process further cumbersome. BRAM and soft-logic voltage rails also are not separately considered, so the minimum employed voltage will be limited by the one that violates the timing first.

Finally, recent work in [56] examines the voltage scaling of FPGA BRAMs. The authors

showed that up to 39% of BRAM voltage can be reduced without observing any error. Though it is promising in reducing the power of BRAMs by one order of magnitude, a trial-and-error based approach does not guarantee correct functionality as it is infeasible to examine all the inputs. Moreover, the overall efficiency is limited to power of BRAMs.

Aforementioned techniques are all speculative as they decrease the voltage until observing error. This overlooks errors that emerge gradually due to violating the guardbands (e.g., timing error as a result of degradation [57]) or may arise abruptly due to voltage transients [29]. Recently, work in [29] showed a margin over 36% is needed for voltage transients as a result of load transients, and this margin is already considered in STA tools. Nevertheless, as voltage transients occur infrequently, the speculative voltage scaling methods do not take them into consideration, which will lead to timing violation at certain conditions. Our voltage scaling approach is different from previous studies as we incorporate it in the FPGA design flow by characterizing the resources during FPGA architecting. This eliminates the arduous task of voltage-timing speculation and guarantees timing. Our method precisely considers the correlation of temperature, delay, voltage, and power of resources and separate power rails of soft-fabric and memories, so it yields maximum efficiency by setting the optimal core and BRAM voltages as well as accurately estimating the timing according to the thermal distribution of the blocks. Finally, *over-scaling* of voltages needs timing simulation to observe the impact of timing violations. We enable it by our novel FPGA simulation flow.

2.3 Proposed Method

2.3.1 Preliminary

FPGA flow is different from conventional standard-cell based design of ASICs, thus we first elaborate the setup of experiments used in the rest of the chapter before detailing the proposed method.

• **Power and Delay.** We use circuit-level simulations to obtain the delay and power of FPGA resources. For this end, we use the latest version of COFFE [58] that generates and characterizes an accurate netlist of FPGA resources according to the given architectural description using comprehensive circuit-level HSPICE simulations. Besides the description of the target FPGA architecture, COFFE also requires technology process of transistors, for which we use 22nm predictive technology model (PTM) [59]. COFFE also generates and evaluates the memory blocks of FPGA and has been shown to have a suitable delay and exact area match with commercial FPGAs [60]. Similar to configuration SRAM cells, the core of the memory blocks (i.e., eight-transistor dual-port SRAM cells) is implemented by 22nm high-threshold low-power transistors [61], which throttles their leakage power by two orders of magnitude. As we will show in the rest of this chapter, our simulations show a similar power trend to commercial FPGAs.

COFFE does not model DSP blocks. We thus develop the DSP HDL code based on the Stratix IV description [62] and characterize it using Synopsys Design Compiler with NanGate 45nm open cell library [63]. Then we scale the results to 22nm based on measuring scaling factors of a selected set of cells at 45nm and 22nm technologies. Based on PrimeTime report, the developed DSP consumes 4.6mW at 250MHz, which is comparable to a 28nm DSP that dissipates 5.6mW at the same frequency [64]. To characterize the delay and power of programmable resources at different temperature and voltages, we sweep the parameters of COFFE-generated netlists in HSPICE simulations. For DSP, we create a set of standard-cell libraries using NanGate netlists by the means of Synopsys SiliconSmart so we could characterize the DSP at different operating conditions.

• **FPGA Flow.** We use VTR 7.0 (Verilog-to-Routing) [65] toolchain that enables defining a customized FPGA architecture and place and route the benchmarks. We use FPGA architecture parameters similar to Intel Stratix devices [66, 1] in COFFE and VPR[1] placement and routing, which is summarized in Table 2.1. COFFE uses these parameters to generate SPICE netlist and

[1]VPR (Versatile Place and Route) is the P&R tool in VTR toolchain.

Table 2.1: FPGA architecture parameters used in COFFE

Parameter	Value	Parameter	Value
K	6	SB_{mux} size	12
N	10	CB_{mux} size	64
Channel tracks	240	$local_{mux}$ size	25
Wire segment length	4	V_{core}, V_{bram}	$0.8V, 0.95V$
Cluster global inputs	40	BRAM	$1024 \times 32\,bit$

area and delay report, which are then fed into VTR to place and route the benchmarks. K and N are the size of LUTs and number of logic blocks in a cluster (see Figure 2.1), which are chosen to be 6 and 10 in accordance to Intel devices. Estimating the power consumption of applications (for both thermal simulation and power saving estimation) needs also their signal activity, for which we use ACE 2.0 [67]. We modified VPR to enable timing analysis at different scenarios using the characterized libraries.

• **Thermal Simulation.** We use HotSpot 6.0 [68] for thermal simulations. Inputs of HotSpot are device floorplan, power trace (or average power values), and device configuration parameters. For the floorplan file, we divide the device floorplan into a two-dimensional array of FPGA tiles with the areas reported by COFFE. We assume CLB tiles are square, and the heights of DSP and memory blocks are $4\times$ and $6\times$ of CLB tiles [65]. Number of tiles and location of each tile can be obtained from the placement and routing outputs reported by VPR. Leakage and dynamic power of each tile is obtained based on the current temperature and activities of resources of the tile. Finally, for the HotSpot configuration file, we change the parameters according to validated FPGA parameters in [69]. We adjust the convective resistance to concur with an effective thermal resistance (θ_{JA}) of contemporary FPGAs, i.e., we tune `r_convec` such that when the total power of given power trace is set to 1 Watt, the reported temperature by HotSpot equals θ_{JA}. We examine the efficiency of the proposed technique by using a typical θ_{JA} of $2^{\circ C}/_{W}$ as in today's Intel and Xilinx devices (e.g., Virtex 7 and Stratix V) [64, 70], and a pessimistic thermal resistance of $12^{\circ C}/_{W}$, corresponding to their mid-size devices (such as Spartan-7 or Artix-7) with still airflow.

Table 2.2: Specifications of the benchmarks.

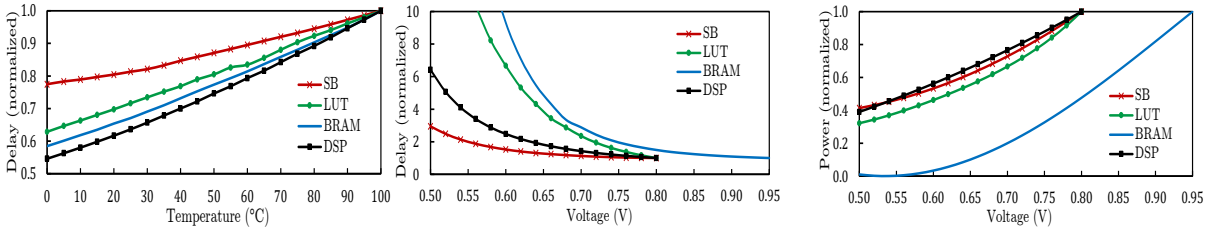
Benchmark	Domain	LUT	BRAM	DSP
boundtop	Ray Tracing	3,132	1	0
ch_intrinsics	Memory Init	459	1	0
LU8PEEng	Math	26,288	45	8
LU32PEEng	Math	87,852	168	32
mcml	Medical Physics	106,348	159	30
mkDelayWorker32B	Packet Processing	6,128	164	0
mkPktMerge	Packet Processing	389	15	0
mkSMAdapter4B	Packet Processing	2,124	5	0
or1200	Soft Processor	3,136	2	1
raygentop	Ray Tracing	2,631	1	9

Benchmarks. We select our benchmarks from VTR repository that belong to a wide variety of applications (vision, math, communication, etc.), contain single- and/or dual-port memory blocks as well as DSP blocks, with an average of over 23,800 6-input LUTs (maximum over 106K). Table 2.2 summarizes the characteristics of these benchmarks.

2.3.2 Proposed Thermal-Aware Voltage Scaling Flow

• **Motivation.** Figure 2.2 gives a perception on how temperature margin can be leveraged for power reduction. This figure is obtained using the experimental setup explained in the previous subsection. Numbers were not in the same range, so we normalized each one to its base value at 100°C and 0.8V for the sake of clear illustration. According to Figure 2.2(a), although FPGA timing analysis reports the worst-case to ensure timing meets in all scenarios [71], in practice, resources have a smaller delay at lower temperatures. For instance, at 40°C, delay of switch box (~~×~~ SB) is 0.85× of its delay at worst-case temperature[2]. This gap can be utilized for voltage reduction. Based on Figure 2.2(b), 0.68V is the point wherein this margin is fully utilized, i.e., delay of 40°C increases by $\frac{1}{0.85\times}$ and becomes equal to delay at worst-case temperature. Eventually, Figure 2.2(c) reveals this 120mV reduction of voltage shrinks the switch box power down by 32%. As mentioned before, memory block comprises of low-threshold transistors, so

[2]We assume an upper-bound of 100°C for junction temperature [62].



(a) Delay-temperature relations. (b) Delay-voltage relations of resources. (c) Power-voltage relations of resources.

Figure 2.2: Different behavior of different FPGA resources under varying temperature and voltages.

uses a different power rail with a voltage higher than datapath (core) transistors. Other non-memory resources show a $\sim V^2$ relation with voltage while BRAM observes a more dramatic power reduction as voltage scales. As is evident from the figure, different resources exhibit different delay behavior as temperature and voltage change. This stems from different sizing of transistors, input slope, output capacitance, etc., as detailed in previous works [72, 49].

We can get several insights from Figure 2.2 and above discussion.

(a) Replica circuits such as ring oscillators used to correlate the temperature/voltage with frequency of ASICs are not a viable solution to track timing because, in FPGAs, CPs are design-dependent and made from different types and count of resources. Apparent from Figure 2.2(a) and (b), designs bounded by routing (SB) have a totally different performance behavior compared to logic (LUT) bounded ones when temperature or voltage changes, so a set of representative paths fails to resemble all paths accurately.

(b) Previous studies rely on worst-case reported paths to check the timing (or to insert error detectors) while lowering the voltage. Nonetheless, from Figure 2.2(a) and (b) we can infer a non-CP path may become CP at lower temperature or voltage. For instance, LUT delay severely increases at lower voltages, so the delay of LUT-bounded paths can exceed originally reported SB-bounded paths. This signifies cautious timing analysis in voltage scaling.

(c) More importantly, even if timing analysis of an FPGA-based design were possible under

arbitrary (T, V) pairs, efficient voltage scaling would be still challenging because, as shown in Figure 2.2(c), resources enjoy differently from voltage reduction. For instance, memory block shows better power saving as voltage scales, while, on the contrary, its delay also increases more under voltage scaling. Thus, for a certain timing headroom, there is a trade-off between power gain and increase of delay (i.e., use up of margin) when there are multiple power rails. This justifies why temperature-voltage-delay correlation and voltage-power libraries are indispensable for a reliable and efficient thermal-aware voltage scaling.

• **Thermal-Aware Voltage Scaling Flow.** Taking the above-mentioned insights into consideration, in the following we present our voltage scaling algorithm as the core of our energy efficiency technique, and then further elaborate it by exemplifying case-studies. Algorithm 1 can be either simply integrated into the current FPGA flow stack (i.e., in the original timing analysis step), or attached as an additional step. In either case, it relies on a pre-characterized library of delay and power, as detailed in Section 2.3.1. If Algorithm 1 is used as an additional step, then it needs to get the post place and route netlist (e.g., in XDL format [73] that is generated for Xilinx FPGAs). Other inputs of the algorithm are maximum temperature surrounding the FPGA board, and sample inputs or activities (we further discuss it later).

FPGA thermal estimation usually relies on a single total power and ambient temperature value to estimate the junction temperature [64], however, we divide the target FPGA into a grid of $m \times n$ tiles, for m and n being the number of FPGA rows and columns. It improves the accuracy of thermal estimation and helps to catch potential hotspot regions, where the blocks have higher delay than the rest of the board, hence need fine-grained timing analysis for both accuracy and efficiency (i.e., to avoid under- or over-estimation of timing). d_{worst} is the delay that conventional one-size-fits-all timing analysis $\mathcal{T}$ of FPGA reports under nominal memory and core voltages and maximum temperature [71] while also considers some margin for reliability issues such as voltage transients [29]. Thus d_{worst} is the target delay that our algorithm attempts to deliver with lower voltages. One drawback of previous voltage scaling approaches [52, 54] is they invade

Algorithm 1: Thermal-Aware Voltage Selection

Input: $netlist$: Placed and routed design
Input: T_{amb}: Ambient temperature
Input: $\overrightarrow{\alpha}$: Input activities / sample inputs

1 $\overrightarrow{T}_{m\times n} = [T_{amb}, \cdots, T_{amb}]$ // m, n : FPGA grid size
2 $\overrightarrow{\Delta T}_{m\times n} = [\infty, \cdots, \infty]$
3 $d_{worst} = \mathcal{T}(netlist, T_{max}, V_{core_{max}}, V_{bram_{max}})$
4 **while** $\|\overrightarrow{\Delta T}\|_\infty > \delta_T$ **do**
5 $\quad \min_{V_{core},\, V_{bram}} \overrightarrow{P}_{lkg}(\overrightarrow{T}, V_{core}, V_{bram}) +$
6 $\quad\quad \overrightarrow{P}_{dyn}(netlist, \overrightarrow{\alpha}, f_{worst}, V_{core}, V_{bram})$
7 $\quad\quad$ s.t. $\mathcal{T}(netlist, \overrightarrow{T}, V_{core}, V_{bram}) \leq d_{worst}$
8 $\quad \overrightarrow{T}_{old} = \overrightarrow{T}$
9 $\quad \overrightarrow{T} = HotSpot(\overrightarrow{P}_{lkg} + \overrightarrow{P}_{dyn})$
10 $\quad \overrightarrow{\Delta T} = \overrightarrow{T} - \overrightarrow{T}_{old}$
11 **return** V_{core}, V_{bram}

this reliability margin when they speculatively reduce the voltage until observing an error in the output, as the error does not show up in regular conditions. The core of the algorithm is a loop where, based on previously obtained temperature for each tile (set to T_{amb} initially), it finds the (V_{core}, V_{bram}) pair that minimizes the power while watches over the delay of candidate pair to not exceed d_{worst}. In the first iteration of the algorithm, it explores all $|V_{core}| \times |V_{bram}|$ pairs. In the next iterations, execution time can be significantly reduced by limiting the search to the boundaries of the previous solution, making subsequent iterations $O(1)$. Note that in both timing analysis and power calculation (lines $5-7$), each tile has its own activity and potentially different temperature, so we use vectors of length $m \times n$ to store the values associated with each tile. Temperature affects the leakage power and delay of the tile resources, while activity affects the dynamic power. Hence, $\overrightarrow{T}$ is passed to $\overrightarrow{P}_{lkg}$ calculation and timing ($\mathcal{T}$) analysis, while $\overrightarrow{\alpha}$ is passed to $\overrightarrow{P}_{dyn}$ to estimate dynamic power at d_{worst} (clock cycle will be always d_{worst}). Finally, the power values are imported in a thermal simulator to update temperatures of tiles. This procedure repeats until reaching a steady-state temperature. For thermal simulation in our

experiments we use HotSpot 6.0 [68], with setup already detailed in Section 2.3.1.

• **Static and Dynamic Implementations.** Static implementation of the proposed technique is straightforward as both core and memory voltages are determined during the configuration of design, whether by incorporating Algorithm 1 in original timing analysis of FPGA, or using it as a post-routing addendum. As the voltages remain fixed in the field operation, the algorithm needs to consider the corner case of the programmed design, i.e., the highest temperature it might reach. A noteworthy point here is that activities of internal nodes of a design do not linearly correspond to activities of primary inputs. In Figure 2.3, when signal activity factor (α) of benchmarks inputs increases from 0.1 to 1, activity of internal nodes (averaged over all 10 benchmarks) increases from 0.05 to ~0.27, which is significantly less than $\alpha = 1$ considered for primary inputs. In addition, in some blocks such as DSP, the increase in activity of primary inputs does not necessarily translate to increase of power. As can be seen from Figure 2.3, DSP power increases by only around 37% when its inputs activities raise from 0.1 to 0.3, then its power saturates until $\alpha \in [0.3, 0.7]$, and declines thereafter. This behavior of power is because the frequently changing inputs offset each other more often (e.g., when both inputs of an XOR function change in a clock, its output remains the same). All in all, by caring for the worst-case input activity, the proposed static scheme guarantees reliable operation at corners without overly pessimistic activity estimation, though the ambient temperature needs to consider maximum possible.

The proposed thermal-aware voltage scaling scheme can be implemented online (dynamic) to avoid pessimistic assumption on the temperature bound. Instead of thermal simulation, online scheme reads the junction temperature using on-board sensors available on all contemporary FPGAs. For instance, Intel devices already contain temperature sensing diodes (TSD) with instantiatable IP cores having their own internal clock source that can output the junction temperature with a resolution of 10 bits in 1,024 clock cycles (i.e., $1ms$) [74]. Therefore, during the configuration of each design, we create a look-up table with temperature T as its keys and

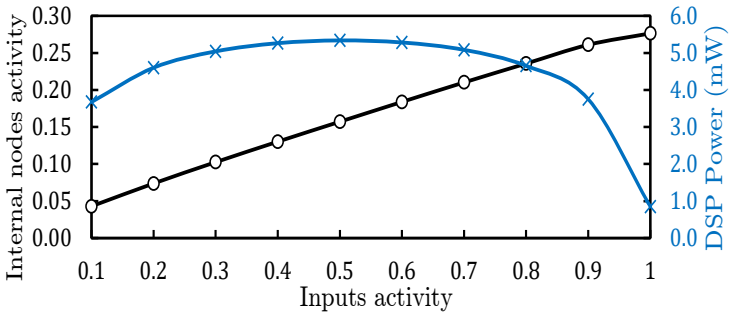
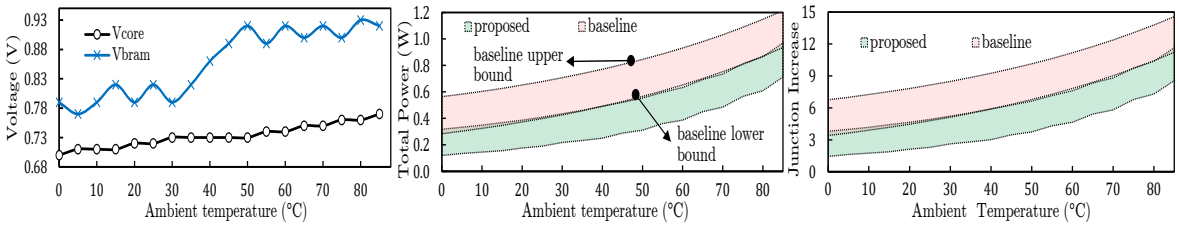


Figure 2.3: Activity of benchmarks nodes for different activities of primary inputs (left/blue), and DSP power at different activities of its inputs (right/red).

(V_{core}, V_{bram}) as the values that minimize the power for that T. Optimal (V_{core}, V_{bram}) of each temperature can be obtained in the same way explained above for the static approach. Reading the temperature with steps of few milliseconds is large-enough to allow on-chip voltage regulators (such as Intel on-the-fly regulators) to adjust the voltage [75], and yet is small enough to avoid temporal heat-up mismatch that takes orders of seconds [76]. The sensed junction temperature can be directly used as VID (voltage identification) for the programmable integrated voltage regulator to adjust the voltage with the pre-loaded values [75]. A thermal margin (e.g., 5°C) might be considered to account for the error of TSDs and potential spatial thermal gradients [77].

• **Case-study.** To elaborate our method, we use `mkDelayWorker` benchmark with $6,128$ LUTs and 164 memory blocks that VPR mapped it to a 92×92 grid device due to its high BRAM demand, with a frequency of 71.6 MHz. Our simulations show the device consumes a leakage power of 0.367mW at 25°C (considering all used and unused resources), while the closest Intel device (Stratix V 5SGSD3) is $1.5\times$ in size with a power of 0.646mW. This $1.76\times$ power ratio is acceptable considering the size difference and more advanced technology we use in our simulations (22nm versus 28nm).

Figure 2.4 shows the results of our static voltage scaling scheme on `mkDelayWorker` benchmark. We assume ambient (near-board) temperature range from 0°C up to 85°C as previous studies have shown that board temperature of datacenter FPGAs can reach up to ~70°C.



(a) Voltage of core and memory power rails.

(b) Total consumed power bounds.

(c) Increase of junction temperature bounds.

Figure 2.4: Outputs of Algorithm 1 for `mkDelayWorker` benchmark under different ambient temperatures.

Figure 2.4(a) shows that, moving from 0°C to 85°C, generally both V_{core} and V_{bram} increase towards their nominal 0.8V and 0.95V values to meet timing in worst-case junction temperature. Small fluctuations of BRAM voltage at certain points is to yield maximum power saving. For instance, at 30°C, $V_{core}, V_{bram} = (0.73, 0.79)$ while at 25°C, $V_{core}, V_{bram} = (0.72, 0.82)$, however, we expected BRAM voltage to be lower for 25°C. This is because actually the 10mV reduction of V_{core} at 25°C is worth the 30mV increase of V_{bram}; as we examined, it resulted in a power of 410mW while experiments showed that the other combination (0.73, 0.79) would consume 420mW ($>$ 410mW). It indicates preciseness of our technique in determining most efficient voltage pairs for a given T_{amb}.

Figure 2.4(b) compares the total power consumption of the proposed technique and baseline. Each curve shows the lower and upper bound of the power, where lower bound corresponds to $\alpha = 0.1$ and upper bound corresponds to maximum dynamic power consumption, i.e., $\alpha = 1.0$. As explained above and showed in Figure 2.3, power does not increase linearly with activity (i.e., upper bound of power is not $10\times$ of lower bound) because leakage power is independent of activity and also activities of internal nodes are not linearly correlated with primary inputs activity. As expected, lower temperatures have more power saving as there is more margin to reduce voltages. Also, although the proposed method optimizes the voltages according to worst-case activity, our method still significantly improves power when activity is low. Note that

the baseline has fixed voltages; however, it also consumes less leakage power in lower temperature so its total power also reduces in lower temperatures. It is noteworthy that in our experiments we observed the leakage power has an exponential relation of $e^{0.015T}$ with temperature, which is comparative to $e^{0.017T}$ we derived for Intel devices [70]. The junction temperature of baseline exceeded 100°C when ambient temperature reaches 85°C. Thus, Figure 2.4(b) and (c) are limited to 85°C.

Finally, Figure 2.4(c) shows the upper ($\alpha = 1.0$) and lower ($\alpha = 0.1$) bounds of increase in junction temperature of device tiles for different ambient temperatures (and corresponding voltages). Higher activity consumes more dynamic power hence has a higher impact on temperature. There is a close correlation between Figure 2.4(b) and (c) as steady-state junction temperature is correlated to total power, especially when design activity is uniform. Please note that overlapping (almost) of lower bound of the proposed method with upper bound of the baseline is haphazard and specific to this benchmark.

• **Algorithm Runtime.** For all of our benchmarks, the flow converges in less than 6 iterations. At low T_{amb} values, due to weak temperature-leakage feedback, the algorithm converges in 2–3 iterations. On a typical desktop system, the first iteration takes less than 12 seconds, and in subsequent iterations the algorithm limits the search space to the boundary of the current solution, making each iteration less than 4 seconds. Thermal simulation takes ~2.5 seconds of each iteration. Table 2.3 shows the details of the static voltage scaling algorithm. At first round, voltages are set to $(0.74, 0.92)$. The resultant power increases the temperature by 5.82° C, which increases the delay (tightens the margin) and leakage. Thus, the second iteration changes the voltages to $(0.75, 0.90)$ for timing closure. The increase of temperature in the first iteration also considerably increases the (leakage) power, from 485mW to 558mW. The temperature then starts converging; hence the subsequent voltage and power changes are insignificant.

• **Discussion.** We do not change the voltages of other power rails such as auxiliary supply voltage (V_{aux}) and I/O voltage (V_{io}) as they enable interfacing with other devices and have

Table 2.3: Iterations of Algorithm 1 on `mkDelayWorker` at $T_{amb} = 60°$ C.

Iter.	$V_{core}(mV)$	$V_{bram}(mV)$	Power (mW)	$T_{junct}(°C)$	Time (s)
1	740	920	485	65.82	10.9
2	750	900	558	66.69	3.1
3	750	910	564	66.76	3.1
4	750	910	564	66.77	3.1
5	750	910	564	66.77	3.1

relatively low power consumption. We also do not touch the voltage of configuration SRAM cells as they use high-threshold mid-oxide transistors with two orders of magnitude less leakage [61]. In addition, we observed that reducing the voltage of SRAM cells causes voltage drop in pass-gate based multiplexer structure of resources, which increases the leakage power of buffers due to non-ideal voltage at their input.

2.3.3 Proposed Thermal-Aware Energy Optimization Flow

While performance is the major concern of high-end FPGA designs, total *energy* usage is a primary concern of battery-backed and IoT applications. The goal of optimal energy exploration is to find the voltage(s) V_{opt} and clock period d_{opt}, for which $E(V_{opt}, d_{opt}) = P(V_{opt}, d_{opt}) \times d_{opt}$ is minimum, where $E(V_{opt}, d_{opt})$ is the design energy consumption rate operating with V_{opt} and clock d_{opt}. To obtain the V_{opt} and clock period d_{opt}, clearly, the operating voltage V_{opt} must be able to deliver the clock period of d_{opt} to avoid timing violations. Second, the design must operate with maximum possible frequency for the given voltage. Otherwise, if the clock period is set to $\alpha \cdot d_{opt}$ ($\alpha > 1$), total energy becomes:

$$E(V_{opt}, \alpha \cdot d_{opt}) = \big(P_{lkg}(V_{opt}) + \frac{P_{dyn}(V_{opt})}{\alpha}\big) \times \alpha \cdot d_{opt}$$
$$= (\alpha P_{lkg}(V_{opt}) + P_{dyn}(V_{opt}))d_{opt} > (P_{lkg}(V_{opt}) + P_{dyn}(V_{opt}))d_{opt}$$

That is, scaling the clock by α scales the dynamic power by $\frac{1}{\alpha}$, but since the execution time also scales by α, the total consumed dynamic energy remains the same. Nonetheless, the leakage

Algorithm 2: Thermal-Aware Energy Optimization

Input: $netlist$: Placed and routed design
Input: T_{amb}: Ambient temperature
Input: $\overrightarrow{\alpha}$: Input activities / sample inputs

1 $E_{min} = \infty$
2 **for** $\forall\, V_{core}$, $\forall\, V_{bram}$ **do**
3 $\quad \overrightarrow{T}_{n\times n} = [T_{amb}, \cdots, T_{amb}]$
4 $\quad \overrightarrow{\Delta T}_{n\times n} = [\infty, \cdots, \infty]$
5 $\quad$ **while** $\|\overrightarrow{\Delta T}\|_\infty > \delta_T$ **do**
6 $\quad\quad d_{max} = \mathcal{T}(netlist, \overrightarrow{T}, V_{core}, V_{bram})$
7 $\quad\quad \overrightarrow{P}_{total} = \overrightarrow{P}_{lkg}(\overrightarrow{T}, V_{core}, V_{bram}) +$
8 $\quad\quad\quad\quad \overrightarrow{P}_{dyn}(netlist, \overrightarrow{\alpha}, d_{max}, V_{core}, V_{bram})$
9 $\quad\quad \overrightarrow{T}_{old} = \overrightarrow{T}$
10 $\quad\quad \overrightarrow{T} = HotSpot(\overrightarrow{P}_{lkg} + \overrightarrow{P}_{dyn})$
11 $\quad\quad \overrightarrow{\Delta T} = \overrightarrow{T} - \overrightarrow{T}_{old}$
12 $\quad$ **if** $d_{max} \times \sum_i \overrightarrow{P_i} < E_{min}$ **then**
13 $\quad\quad E_{min} = d_{max} \times \sum_i \overrightarrow{P_i}$
14 $\quad\quad V_{core_{min}} = V_{core}$
15 $\quad\quad V_{bram_{min}} = V_{bram}$
16 **return** $V_{core_{min}}, V_{bram_{min}}$

power is independent of the clock period. Thus, the leakage energy scales by α. Therefore, for a given voltage (which we aim to find the best one), the clock *period* needs to be minimum possible to minimize total energy.

That being said, we derive the new Algorithm 2 that looks for the (V_{core}, V_{bram}) pair that achieves minimum power-delay product as the energy metric whilst also exploits the temperature headroom for further efficiency. Clearly, having the temperature-delay-voltage and voltage-power characterization is vital for the reliability and efficiency of the proposed flow. Having Algorithm 1 already explained, understanding the Algorithm 2 is straightforward. Essentially, it looks for all (V_{core}, V_{bram}) pairs, and as reasoned above, finds the maximum frequency considering thermal margin. In contrast with the voltage scaling flow, Algorithm 2 exploits the available thermal margin to maximize the frequency for a candidate voltage, rather than lowering the voltage for a fixed frequency. Thermal simulation (line 10) is again crucial as changing the frequency (line 6)

changes the power and hence temperature.

Algorithm 1 was performing thermal simulation for the best (V_{core}, V_{bram}) at each iteration and we observed that, in the worst case, it converges in less than eight iterations. However, Algorithm 2 needs to explore all $|V_{core}| \times |V_{bram}|$ combinations and perform several thermal simulations under each. It could take up to several hours for large benchmarks. We enhanced it by, first, skipping a (V_{core}, V_{bram}) combination if its energy in the initial loop (i.e., before involving temperature-delay feedback of line 10) was larger than the already found optimum. In addition, we also considered a small temperature margin of $0.1°C$, so we could avoid the thermal simulation of cases with power within $\frac{0.1}{\theta_{JA}}$ range of a previously obtained case. These optimizations reduced the average runtime on benchmarks by two-order of magnitude (from 72 minutes to 49 seconds) with virtually no impact on the solution.

2.3.4 Timing-Speculative Voltage Over-Scaling

Timing speculation has been shown to provide opportunistic power reduction in applications that can inherently tolerate a certain amount of error. Examples are: (a) image processing circuits such as DCT/IDCT where small drop of PSNR (Peak Signal to Noise Ratio) might not be perceived by human [78]. (b) Deep Neural Networks (DNNs) because of their pooling layers that filter out a significant portion of intermediate results and also the stochastic nature of the gradient descent [79] (c) light-weight alternatives of DNNs such as the brain-inspired computing that performs the main machine learning applications by using inexpensive operations on hypervectors that can bear a certain amount of inaccuracy [80], etc. Timing-speculative voltage over-scaling is orthogonal to the thermal-aware voltage reduction with the opportunity of violating the timing for more significant power saving.

Timing-speculative voltage scaling requires post P&R simulation of the targeted design with the timing data at the scaled voltage to estimate the incurred inaccuracy. In standard-cell design approaches, timing speculation can be realized by generating the same cell library under

scaled voltages or providing multiple operating condition modes at the same library. Thus, the synthesized design can be simulated using the new timing data. Nonetheless, to the best our knowledge, there is no FPGA framework for post place and route (timing) simulation, particularly under varying voltages.

By taking advantage of our characterization libraries, we build our simulation framework upon the Verilog-to-Routing (VTR) [65] project explained in Section 2.3.1. Figure 2.5 demonstrates the proposed post P&R simulation framework. The flow begins with synthesis (using ODIN [81]) and technology mapping (using Berkeley ABC [82]) of the HDL description to BLIF format that can be imported to the VPR P&R tool [65]. Architectural description of the target FPGA needed by VPR can be hand-written or auto-generated by COFFE [58]. VPR generates a `.net` file that describes the placement information of resources, and a `.route` file that provides the routing information of design nets. Finally, we developed a Python-based tool to analyze VPR outputs and instantiate the FPGA's utilized components using primitives similar to Xilinx syntax in Verilog HDL. The generated file is augmented with the delay of the resources obtained by parsing the VPR outputs. It is noteworthy that all configurable multiplexers (SBs, CBs, etc.) are programmed in a place and routed design, so we could simplify their functionality as a buffer just to incorporate their delay information. VPR's placement output (`.net`) does not include a functional description. Thus, we retrieve the configuration of LUTs from the technology-mapped BLIF. Similarly, BLIF files do not contain BRAMs initialization, so we read them back from the original HDL file. To make the voltage over-scaling efficient, we utilize the proposed thermal-aware voltage scaling as follows. For a given timing rate (e.g., $1.1\times$ of original clock), we change the timing condition of Algorithm 1 (line 7) to meet the new constraint ($1.1\times$ of d_{worst}). Hence, the obtained over-scaled voltages are optimal for that allowed amount of violation. We repeat it for different timing violation rates. In Section 2.4 we report the additional power saving granted by speculative voltage-scaling.

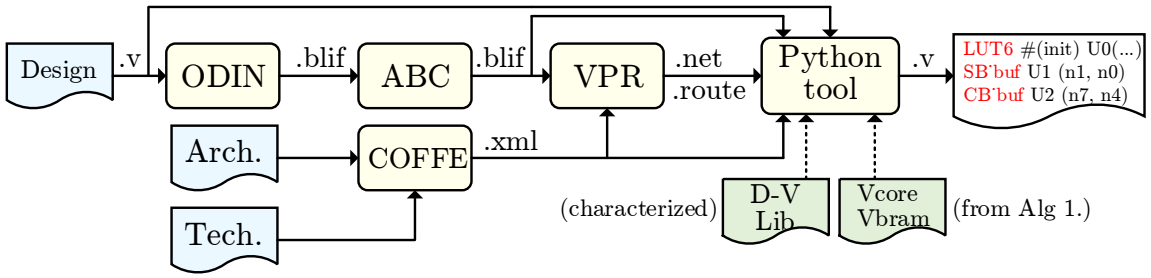
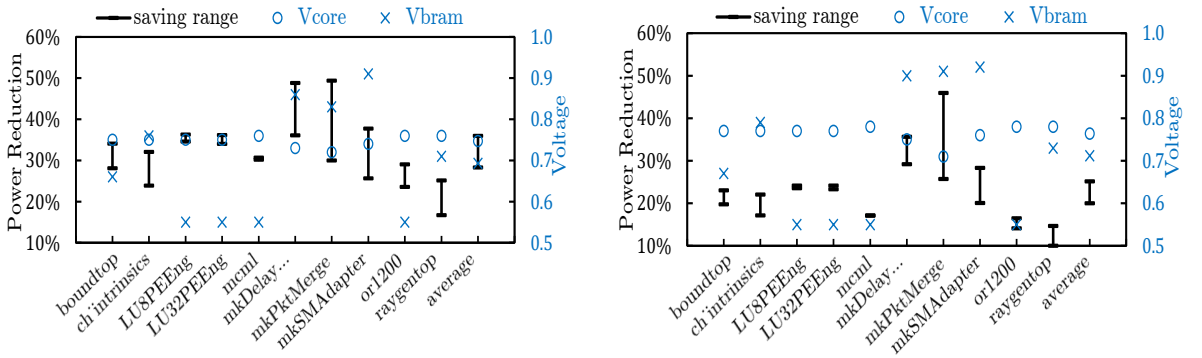


Figure 2.5: The proposed simulation flow for FPGA-mapped applications, enabling speculative voltage scaling.



(a) Range of power reduction at 40°C (left axis) and corresponding core and memory voltages of each benchmark (right axis).

(b) Range of power reduction at 65°C (left axis) and corresponding core and memory voltages of each benchmark (right axis).

Figure 2.6: Power reduction and voltages for 40°C and 65°C board temperatures.

2.4 Experimental Results

• **Power Reduction.** Figure 2.6 demonstrates the power reduction using the proposed flow of Algorithm 1. The flow finds the optimum core and memory voltage pair (V_{core} and V_{bram}) assuming highest inputs activity (α) to guarantee it satisfies temperature corners. In practice, however, the input activity range might be lower. Therefore, for the obtained optimal voltages, we assumed a varying activity $\alpha \in [0.1, 1.0]$ and calculated the power reduction for the entire range. Therefore, Figure 2.6 demonstrates a range of power saving. The right axis of this figure also shows the optimal V_{core} and V_{bram} voltages for each benchmark. In several benchmarks, the paths containing memories were significantly shorter than critical paths. For instance, in `LU8PEEng`, the

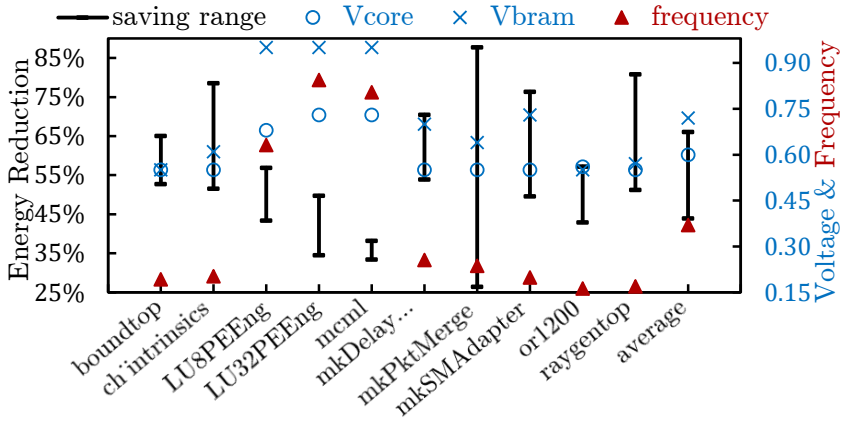


Figure 2.7: Range of energy savings at 65°C (left axis), and corresponding optimal voltage values and frequency ratio (right axis).

critical path is 21× longer than the longest BRAM path. For these paths, V_{bram} is reduced down to 0.55V, which we set as the lowest voltage level before device crashes [56]. In Figure 2.6(a) we considered a device operating at $T_{amb} = 40°C$ with $\theta_{JA} = 12^{°C}/_{W}$, and in Figure 2.6(b) as considered more high-end device operating at 65°C with $\theta_{JA} = 2^{°C}/_{W}$ (see Section 2.3.1 for details of experiments). The opportunity of power saving reduces in higher temperatures. At 40°C, the average power saving of 10 benchmarks is $28.3\% - 36.0\%$ (depends on activity), while at 65°C it becomes $20.0\% - 25.0\%$. We observed up to 9.2°C increase in the junction temperature of the baseline, which reduced to 5.9°C in the proposed method due to consuming less power. Benchmarks have different power reduction and optimal voltages based on the resources on critical paths (that determine the voltage scaling limit), used resources, the activity of nodes, etc. Comparing Figure 2.6(a) and (b) also reveals how differently the voltages of benchmarks need to be adjusted moving from 40°C to 65°C: `raygentop` needs boosting both voltages by 20mV, `or1200` needs only +20mV of core rail, and `mkPktMerge` needs 80mV increase of memory rail with 10mV reduction of core voltage, suggesting the necessity of dynamic implementation (see Section 2.3.2) for best efficacy.

• **Energy Reduction.** Figure 2.7 shows the range of energy reduction (left axis) of each benchmark using the proposed energy optimization flow at 65°C. The points (×, ○, ▲) correspond

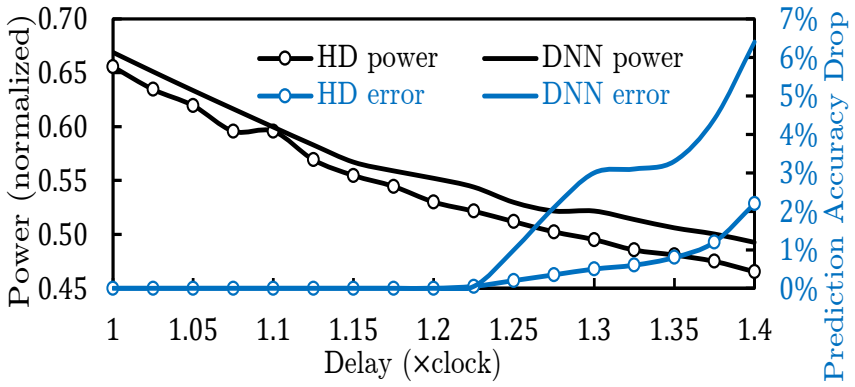


Figure 2.8: Power reduction (left axis) and error increase (right axis) under voltage over-scaling. X-axis shows violation of critical path delay. T_{amb} is 40°C.

to the right axis and show the optimal voltage values and frequency ratio. Remember that the goal of our energy minimization flow was to find out the minimum energy consumption point (power-delay product) by compromising the delay and power, so the delay has been increased by $\frac{1}{0.37} = 2.7\times$ while the overall consumed energy is improved by $44\% - 66\%$ depending on input activities. There are obvious differences with optimal points of power and energy reduction flows. Unlike the power flow, here, V_{bram} of above-mentioned benchmarks (e.g., `LU8PEEng`) have not been shrunk down to 0.55V because the delay of their critical paths is also increased, hence the memory voltage cannot be freely reduced. The ranges of energy savings are also more stretched (e.g., in `mkPktMerge`) because in higher activities, memory dynamic energy becomes the dominant contributor to total energy consumption, hence, throttling its energy becomes worthwhile even considering the increased delay. As it can be seen in the figure, its V_{core} is reduced to 0.64V while in power reduction flow (Figure 2.6(b)) it could be reduced to 0.91V because the clock delay must have been remained fixed.

• **Speculative Voltage Over-Scaling.** We chose `LeNet` [83] as a classic CNN for hand-written digit recognition and implemented as a systolic array architecture [84]. Its relatively small size makes the timing simulation computationally tractable. We also selected another machine learning algorithm based on computing with hyperdimensional (`HD`) [85] vectors to

detect two face/non-face classes among 10,000 web faces of a face detection dataset (FACE) from Caltech [86]. Figure 2.8 shows the result of thermal-aware voltage over-scaling. Initially CP delay is 1× of original clock, meaning that no timing violation is allowed and the ∼34% power reduction is because of thermal-aware voltage scaling. Thereafter, we allow up to 40% violation of CP delay, where accuracy drop becomes noticeable at 1.2× of the original clock. This tolerance is because DNNs are intrinsically error-tolerant (e.g., allow quantization of weights down to 3 bits in the LeNet [87]). Similarly, previous studies of HD have shown an accuracy drop of merely 4% when up to 30% of the vectors bits are flipped (i.e., noisy) [80] mainly because orthogonality of vectors, making them discernible under error. Based on Figure 2.8, when voltage over-scaling increases the CP delay to 1.35× of the clock period, errors start spiking. At this point, by respectively 3% and 0.5% accuracy drop, LeNet and HD powers are reduced by 48% and 50%, which means additional 15% and 16% improvement compared to our original voltage-scaling (with ∼34% improvement for both) in which CP delay does not exceed clock period.

2.5 Conclusion

In this chapter, we introduced the inefficiencies caused by the pessimistic timing margin employed on FPGA-based designs. We also discussed why coping with this issue on FPGAs, due to the reconfigurability of these devices and using various types of resources, is significantly more challenging compared to ASICs. We then proposed systematic power and energy optimization techniques by characterizing FPGA resources to utilize the thermal headroom. While keeping the performance intact, the voltage scaling flow determines the optimal voltages of designs based on their thermal distribution and can be implemented statically with fixed voltage(s) or dynamically using programmable voltage regulators at highly varying ambient temperatures. Our proposed energy optimization flow compromises the delay and power to seek the optimal point that minimizes total consumed energy. Finally, we proposed a timing simulation framework

that provides further power saving opportunities by making the impact of voltage over-scaling observable.

FPGA devices are nowadays popular on both data center and edge applications. Hence, addressing the energy efficiency of these devices impacts a wide variety of application domains. Nevertheless, nowadays, more and more applications rely on heavy machine learning algorithms such as DNNs to gain useful insight from data. While such machine learning techniques can produce state-of-the-art results, they entail substantial memory and compute requirements that exceed the energy, power, and resource (area) overhead of many edge applications. In the next chapter, we discuss how we can bring the learning capability to the edge, particularly in domains with extremely tight resource and energy constraints.

Chapter 2, in part, is a reprint of the material as it appears in "FPGA Energy Efficiency by Leveraging Thermal Margin" by Behnam Khaleghi, Sahand Salamat, Mohsen Imani, and Tajana Rosing which appears in IEEE International Conference on Computer Design (ICCD), 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Efficient FPGA Implementation of Hyperdimensional Computing by Approximation

In the previous chapter, we proposed a systematic approach to lower the power and energy consumption of FPGA devices, which impacts a wide variety of application domains in different levels of the computing stack, from edge to cloud. However, nowadays, many applications rely on machine learning algorithms to gain useful insight from data. The efficacy of these algorithms is usually correlated with their compute and power requirements. In this context, DNNs often stand out as they can produce state-of-the-art results. Nonetheless, the resource requirement and energy consumption of DNNs are far beyond the limitations of many edge devices. Therefore, in the rest of the dissertation, we focus on a new and efficient learning paradigm, hyperdimensional computing (HDC), and propose efficient hardware and algorithm designs for HDC to bring affordable learning to the edge. To this end, in this chapter, we introduce HD computing and

propose efficient FPGA implementations. Particularly, we propose approximate yet accurate architectures that reduce resource utilization by more than 80% and facilitates implementing HDC on very low-end FPGAs.

3.1 Introduction

The growing number of edge devices creates an exploding amount of data every moment. Applications running on edge devices nowadays rely more and more on machine learning (ML) algorithms to generate useful insights from data. While modern machine learning techniques such as deep neural networks can produce state-of-the-art results, they entail substantial memory and compute requirements that can exceed the resources available on most of edge devices. Thus, there is a pressing need to develop novel machine learning techniques that provide accuracy and flexibility while meeting the tight resource constraints of edge devices.

Hyperdimensional computing – HD for short – is an emerging paradigm for machine learning based on evidence from the neuroscience community that the brain "computes" on high-dimensional, distributed, representations of data [30, 88, 89, 90, 91, 32, 31]. In HD, the primitive units of computation are high-dimensional vectors of length d_{hv} sampled randomly from the uniform distribution over the binary cube $\{\pm 1\}^{d_{hv}}$. Typical values of d_{hv} are in the range 5-10,000. Because of their high-dimensionality, any randomly chosen pair of points will be approximately orthogonal (that is, their inner product will be approximately zero). A useful consequence of this is that sets can be encoded simply by summing (or "bundling") together their constituent vectors. For any collection of vectors $\mathbf{P}, \mathbf{Q}, \mathbf{V}$ their element-wise sum $\mathbf{S} = \mathbf{P} + \mathbf{Q} + \mathbf{V}$ is, in expectation, closer to $\mathbf{P}, \mathbf{Q}$ and $\mathbf{V}$ than any other randomly chosen vector in the space.

Given HD representations of data, this provides a simple classification scheme: we simply take the data points corresponding to a particular class and superimpose them into a single representation for the set. Then, given a new piece of data for which the correct class label is

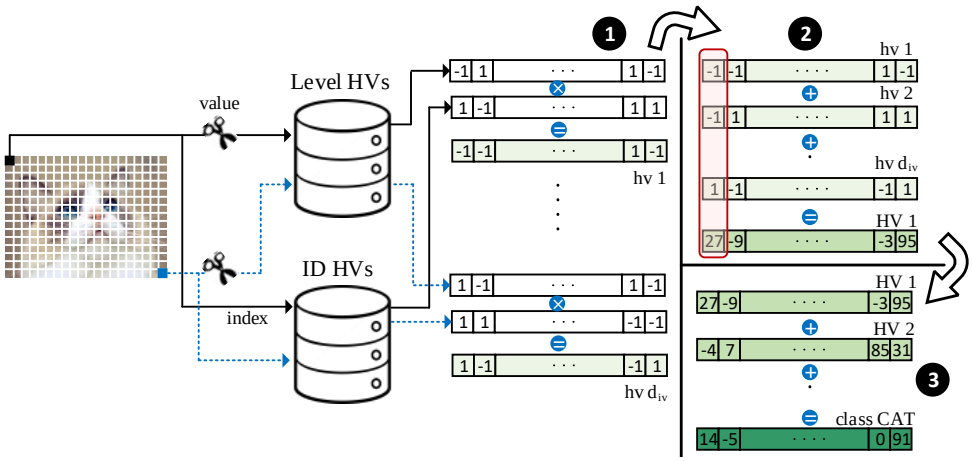


Figure 3.1: Encoding and training in HD.

unknown, we compute the similarity with the hypervectors representing each class and return the label corresponding to the most similar one. More formally, suppose we are given a set of labeled data $\mathcal{X} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ where $\mathbf{x} \in \mathbb{R}^{d_{iv}}$ corresponds to an observation in low-dimensional space and $y \in \mathcal{C}$ is a categorical variable indicating the class to which a particular $\mathbf{x}$ belongs. In general, HD classification proceeds by generating a set of "class hypervectors" which represent the training data corresponding to each class. Then, given a piece of data for which we do not know the correct label – the "query" – we simply compute the similarity between the query and each class hypervector and return the label corresponding to the most similar. This process is illustrated in Figure 3.1.

Suppose we wish to generate the class hypervector corresponding to some class $k \in \mathcal{C}$. The prototype can be generated simply by superimposing (also called "bundling" in the literature) the HD-encoded representation of the training data corresponding to that particular class [30, 92]:

$$\mathbf{C}_k = \sum_{i \text{ s.t. } y_i = k} \text{enc}(\mathbf{x}_i) \tag{3.1}$$

where $\text{enc} : \mathbb{R}^{d_{iv}} \to \{\pm 1\}^{d_{hv}}$ is some *encoding function* which maps a low-dimensional signal to a binary HD representation. Then, given some piece of "query" data $\mathbf{x}_q$ for which we *do not* know

the correct label we simple return the predicted label as:

$$k^\star = \underset{k \in C}{\operatorname{argmax}}\ \delta(\text{enc}(\mathbf{x}_q), \mathbf{C}_k) \tag{3.2}$$

where δ is an appropriate similarity metric. Common choices for δ include the inner-product/cosine distance – appropriate for integer or real valued encoding schemes – and the hamming distance – appropriate for binary HD representations. This phase is commonly referred to in literature as "associative search". Despite the simplicity of this "learning" scheme, HD computing has been successfully applied to a number of practical problems in the literature ranging from optimizing the performance of web-browsers [93], to DNA sequence alignment [94, 95], bio-signal processing [96], robotics [97, 98], and privacy-preserved federated learning [42].

The primary appeal of HD computing lies in its amenability to implementation in modern hardware accelerators. Because the HD representations (e.g. $\phi(\mathbf{x})$) are simply long Boolean vectors, they can be processed extremely efficiently in highly parallel platforms like GPUs, FPGAs and PIM architectures. The principal challenge of HD computing – and the focus of this dissertation – lies in designing good encoding schemes which (1) represent the data in a format suitable for learning and (2) are efficient to implement in hardware. In general, the encoding phase is the most expensive stage in the HD learning pipeline – in some cases taking up to $10\times$ longer than training or prediction [99]. Existing encoding methods require generating hypervectors in full integer-precision and then *ex-post* quantizing to $\{\pm 1\}$. While this accelerates the associative search phase, it does not address encoding which is the primary source of inefficiency.

In this work, we propose novel techniques, dubbed SHEAR*er* (highly-efficient hyperdimensional computing by software-hardware enabled Multifold approximation) to compute the encodings in an approximate manner that saves a substantial amount of resources with an insignificant impact on accuracy. Of independent interest is our novel FPGA implementation that achieves striking performance through massive parallelism with low power consumption.

Approximate encodings entail models to be trained in a similar approximate fashion. Thus we also develop a software emulation to enable users to train desired HD models. Our software framework enables users to explore the tradeoff between the degree of approximation, accuracy, and resource utilization (hence power consumption) by generating a pre-compiled library that correlates approximation schemes and FPGA resource utilization and power consumption. We show our procedure leads to performance improvement of 104,904× (15.7×) and energy savings of up to 56,044× (301×) compared to state-of-the-art encoding methods implemented on Raspberry Pi 3 (GeForce GTX 1080 Ti).

3.2 Background and Motivation

3.2.1 HD Encoding Algorithms

The literature has proposed a number of encoding methods for the multitude of data types which arise in practical learning settings. We here focus on a method from [30, 36, 92] which we refer to as "ID-vector" based encoding. This encoding method is widely used (see for instance: [36, 96, 100, 101]) and works well on both discrete and continuous data. We focus the discussion on continuous data as discrete data is a simple extension.

Suppose we wish to encode some set of vectors $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ where $\mathbf{x}_i$ is supported on some compact subset of $\mathbb{R}^{d_{iv}}$. To begin, we first quantize the domain of each feature into a set of L discrete values $\mathcal{L} = \{l_i\}_{i=1}^{L}$ and assign each $l_i \in \mathcal{L}$ a codeword $\mathbf{L}_i \in \{\pm 1\}^{d_{hv}}$. To preserve the *ordinal relationship* between the quantizer bins (the l_i), we wish the similarity between the codewords $\mathbf{L}_i, \mathbf{L}_j$ to be inversely proportional to distance between the corresponding quantization bins; e.g. $\delta(\mathbf{L}_i, \mathbf{L}_j) \propto |l_i - l_j|^{-1}$. To enforce this property we generate the codeword $\mathbf{L}_1$ corresponding to the minimal quantizer bin l_1 by sampling randomly from $\{\pm 1\}^{d_{hv}}$. The codeword for the second bin is generated by flipping $\frac{d_{hv}}{2 \cdot L}$ random coordinates in L_1. The codeword for the third bin is generated analogously from L_2 and so on. Thus, the codewords for the

minimal and maximal bins are orthogonal and $\delta(\mathbf{L}_i, \mathbf{L}_j)$ decays as $|j-i|$ increases. This scheme is appropriate for quantizers with linearly spaced bins – however, it can be extended to variable bin-width quantizers.

To complete the description of encoding, let $q(x_i)$ be a function which returns the appropriate codeword $\mathbf{L} \in \mathcal{L}$ for a component $x_i \in \mathbf{x}$. Then encoding proceeds as follows:

$$\mathbf{X} = \sum_{j=i}^{d_{iv}} q(x_i) \otimes \mathbf{P}_i \tag{3.3}$$

Where $\mathbf{P}_i$ is a "position hypervector" which encodes the index of the feature value (e.g. $i \in \{1, .., d_{iv}\}$) and $\otimes$ is a "binding" operation which is typically taken to be `XOR`.

3.2.2 Motivation

While the basic operations of HD are simple, they are numerous due to its high-dimensional nature. Prior work has proposed varied algorithmic and hardware innovations to tackle the computational challenges of HD. Acceleration in hardware has typically focused on FPGAs [102, 103, 104] or ASIC-ish accelerators [40, 105]. FPGA-based implementations provide a high degree of parallelism and bit-level granularity of operations that significantly improves the performance and effective utilization of resources. Furthermore, FPGAs are advantageous over more specialized ASICs as they allow for easy customization of model parameters such as lengths of hypervectors (d_{hv}) and input-vectors (d_{iv}) along with the number of quantization levels. This flexibility is important as learning applications are heterogeneous in practice. Accordingly, we here focus on an FPGA based implementation but emphasize our techniques are generic and can be integrated with ASIC- [105] and processor-based [40] implementations.

As noted in the preceding section, the element-wise sum is a critical operation in the encoding pipeline. Thus, popcount operations play a critical role in determining the efficiency of HD computing. Figure 3.2(a) shows a popular tree-based implementation of popcount that adds

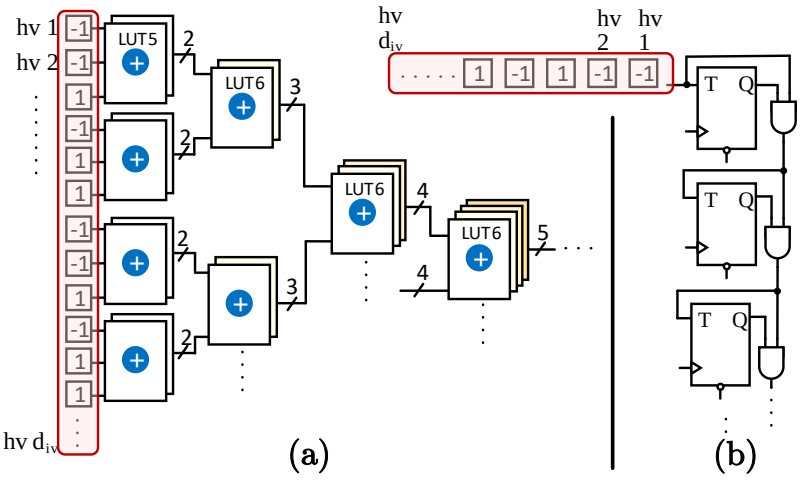


Figure 3.2: (a) Adder-tree and (b) counter-based implementation of popcount. ⊕ denotes add operation.

d_{iv} binary bits (note that we can replace '-1's by 0 in the hardware). Each six-input look-up table (LUT-6) of conventional FPGAs consists of two LUT-5. Hence, we can implement the first stage of the tree using $\frac{d_{iv}}{3}$ of three-port one-bit adders. Each subsequent stage comprises two-port k-bit adders where k increases by one at each stage, while the number of adders per stage decreases by a factor of $\frac{1}{2}$. A n-bit adder requires n LUT-6. Thus, the number of LUT-6 for a d_{iv}-input popcount can be formulated as Equation (3.4).

$$n_{\text{LUT6}}(\text{adder-tree}) = \sum_{i=1}^{\log d_{iv}} \frac{d_{iv}}{3} \times \frac{i}{2^{i-1}} \simeq \frac{4}{3} d_{iv} \tag{3.4}$$

HD operations can be parallelized at the granularity of a single coordinate in each hypervector: all dimensions of the encoding hypervector and associative search can be computed in parallel. Nonetheless, Equation (3.4) reveals that the popcount module for a popular benchmark dataset [106] with 617 features per input requires ∼820 LUTs. This limits a mid-size low-power FPGA with ∼50K LUTs [107] to generate only ∼60 encoding dimension per cycle (out of $d_{hv} \simeq 5{,}000$).

To save resources, [104] and [105] suggest using counters to implement the popcount for

each dimension of encoding, as shown in Figure 3.2 (b). Although this seems more compact, in practice, it is less efficient than an adder-tree implementation: the counter-based implementation needs "$\log d_{iv}$" LUTs per dimension, with a per-dimension latency of d_{iv} cycles, while adder-trees require O($\frac{4}{3}d_{iv}$) LUTs per dimension with a per-dimension throughput of one cycle, so for a given amount of resources, the conventional adder-tree is $\frac{3}{4}\log d_{iv}\times$ more performance-efficient.

Work in [102] and [103] quantize the dimensions of encoding and class hypervectors which eliminates DSP modules (or large number of cascaded LUTs) that are conventionally used for the associative search stage, since, through quantization, inner product for cosine similarity will be replaced by popcount operations in case of binary quantization, or lower-bit multiplications. The resulting improvement is minor because the quantization is applied *after* full-bit encoding. Furthermore, the multipliers of the associative search stage have input widths of w_{enc} (from encoding dimensions) and w_{class} (from class dimensions), so each one needs O($w_{enc} \times w_{class}$) LUTs. Pessimistically assuming bit-widths up to $w_{enc} = w_{class} = 16$, an extreme binary quantization can eliminate 256 LUTs required for multiplication. However, the savings are again modest at best in practice: on the benchmark dataset mentioned previously, only $\frac{w_{enc}\times w_{class}}{w_{enc}\times w_{class}+\frac{4}{3}d_{iv}} \simeq 23\%$. Therefore, we target the popcount portion that contributes to the more significant part of resources. Indeed, *ex-post* quantizing of encoding hypervectors can be orthogonal to our technique for further improvement.

3.3 Proposed Method: SHEAR*er*

3.3.1 Approximate Encoding

In the previous section, we explained prior work that applies quantization after obtaining the encoding hypervector in full bit-width. As noted there, while this approach is simple it only accelerates the associative search phase and does not improve encoding - which is often the principal bottleneck. Because the HD representation of data entails substantial redundancy

and information is uniformly distributed over a large number of bits, it is robust to bit-level errors: flipping 10% of hypervectors' bits shows virtually zero accuracy drop, while 30% bit-error impairs the accuracy by a mere 4% [80]. We leverage such resilience to improve the resource utilization through approximate encoding, as shown in Figure 3.3. In the following, we discuss each technique in greater detail and estimate its resource usage.

(1) Local majority. From Equation (3.4) we can observe that the number of resources (in terms of LUT-6) of the exact adder-tree to see that the complexity encoding each dimension linearly depends on the number of data features, d_{iv}. We, therefore, aim to reduce the number of inputs to the primary adder-tree by sub-sampling using the majority function so as to shrink the tree inputs while (approximately) extracting the information contained in the input. Note that, here, 'inputs' are the binary dimensions of the level hypervectors (see Figure 3.1 ❷ and Figure 3.2). As shown in Figure 3.3(a), each LUT-6 is configured to return the majority of its six input bits. When three out of six inputs are 0/1, we break the tie by designating *all* LUTs that perform majority functions of a specific encoding dimension to deterministically output 0 or 1. We specify this randomly for every dimension (i.e., an entire adder-tree) but it remains fixed for a model during the training and inference. We choose groups of six bits as a single LUT-6 can vote for up to six inputs. Using smaller majority groups diminishes the resource saving, especially taking the majorities adds extra LUTs. Moreover, following the Shannon decomposition, implementing a '$k+1$'-input LUT requires two k-input LUTs (and a two-input multiplexer). Thus, the number of LUTs for majority groups larger than six inputs grows exponentially.

There are $\frac{d_{iv}}{6}$ `MAJ` LUTs in the first stage of Figure 3.3(a), hence the number of inputs for the subsequent adder-tree reduces to $\frac{d_{iv}}{6}$. From Equation (3.4) we also know that a k-input adder-tree requires $\frac{4}{3}k$ LUT-6. Thus, the design of Figure 3.3(a) consumes:

$$\overbrace{\frac{d_{iv}}{6}}^{\texttt{MAJ LUT-6}} + \overbrace{\frac{4}{3}\frac{d_{iv}}{6}}^{\text{adder-tree}} = \frac{7}{18}d_{iv} \text{ LUT-6} \tag{3.5}$$

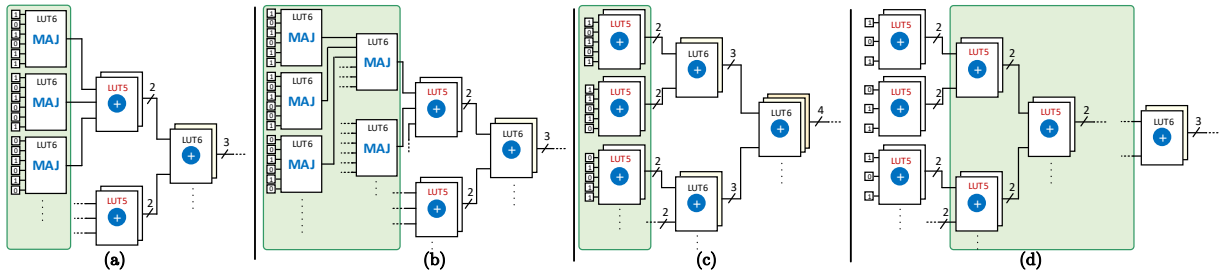


Figure 3.3: Our proposed approximate encoding techniques. MAJ and ⊕ denote majority and addition, respectively.

This uses $1-\frac{7/18}{4/3}=70.8\%$ less LUT resources than an exact adder-tree.

In [103], the authors report an average accuracy loss of 1.6% by *post-hoc* quantizing the encodings to binary. Thus, one might think of repeating the majority functions in the subsequent stages to obtain final one-bit encoding dimensions. Using local majority functions is efficient, but degrades the encoding quality as majority is not associative. In particular, the MAJ LUTs add another layer of approximation by breaking ties. Thus, a so-called MAJ-tree causes considerable accuracy loss. Therefore, in our cascaded-MAJ design in Figure 3.3(b), we limit the MAJ stages to the first two stages. Our cascaded-MAJ utilizes:

$$\overbrace{\frac{d_{iv}}{6}}^{\text{1st stage MAJs}}+\overbrace{\frac{d_{iv}/6}{6}}^{\text{2nd stage MAJs}}+\overbrace{\frac{4}{3}\frac{d_{iv}/6}{6}}^{\text{adder-tree}}=\frac{25}{108}d_{iv}\ \text{LUT-6} \tag{3.6}$$

which saves $1-\frac{25/108}{4/3}=82.6\%$ resources compared to exact encoding. We emphasize that a cascaded all-MAJ popcount needs $\simeq\sum_{i=1}\frac{1}{6^i}=0.2d_{iv}$ LUTs, which saves 85.0% of LUTs. So the two-stage MAJ implementation with 82.6% resource saving is nearly optimal because the first two stages of the exact tree were consuming the most resources.

(2) Input overfeeding. In Figure 3.2(a) we can observe that each LUT-5 pair of the first stage computes $\overline{s_1 s_0}=hv_i+hv_{i+1}+hv_{i+2}$. Since only three (out of five) inputs of them are used, these LUTs left underutilized. With one more input, the output range will be $[0-3]$, which requires three bits (outputs) to represent, so we cannot add more than three bits using two

LUT-5s. However, instead of using the LUT-5s to carry out regular addition, we can supply a pair of LUT-5s with five inputs to perform quantized/truncated addition. For actual outputs (sum of five bits) of 0 or 1, the LUT-5 pair would produce 00 (zero); for 2 or 3 they produce 01 (one), and for 4 or 5 they produce 10 (two). That is one LUT-5 computes the actual carry out of the five bits, and the other computes MSB of the sum. To ensure that the synthesis tool infers a single LUT-6 for each pair, we can directly instantiate LUT primitives. As a LUT-6 comprises a LUT-5 pair (with shared inputs), the number of resources of Figure 3.3(c) is:

$$\sum_{i=1}^{\log d_{iv}} \frac{d_{iv}}{5} \times \frac{i}{2^{i-1}} \simeq \frac{4}{5} d_{iv} \ \text{LUT-6} \tag{3.7}$$

The first stage encompasses $\frac{d_{iv}}{5}$ LUT-6s, and each subsequent stage contains i-bit adders while their count decreases by $\frac{1}{2}\times$ at each stage. Total number of LUTs is reduced by $1 - \frac{4/5}{4/3} = 40\%$ (the same ratio of over-use of inputs). The saving is smaller than the local majority approach but we expect higher accuracy due to intuitively more moderate imposed approximation.

(3) Truncated nodes. Out of $\frac{4}{3} d_{iv}$ LUTs used in an exact adder-tree, d_{iv} (75%) are used in the intermediate adder units. More precisely, following $\frac{i}{2^i}$ ratio (see Equation (3.4)), stages 1–4 of the adder contribute to 25%, 25%, 18.75%, and 12.5% of the total resources, respectively. Note that, although the number of adder units halves at each stage, the area of each one increases linearly. We avoid a blowup of adder sizes by truncating the least significant bit (LSB) of each adder. As demonstrated in Figure 3.3(d), the LSB of the second stage (which is supposed to have three-bit output) is discarded. Thus, instead of using two LUT-6s to compute $\overline{s_2 s_1 s_0} = \overline{a_1 a_0} + \overline{b_1 b_0}$, we can use two LUT-5s (equivalent to one LUT-6) to obtain $\overline{s_2 s_1} = \overline{a_1 a_0} + \overline{b_1 b_0}$, where one LUT-5 computes s_2 and the other produces s_1 using four inputs a_0, a_1, b_0, and b_1. Truncating the output of the second stage consequently decreases the output bit-width of the third stage by one bit as its inputs became two bits. Thus, we can apply the LSB truncating to the third stage to implement it using two LUT-5s, as well. We can apply the same procedure in all the consecutive nodes and

implement them by only two LUT-5s. The output of the first stage is already two bits so we do not modify its original implementation.

We apply truncating to first stages particularly from the left side of Equation 3.4 we can perceive the first five stages that contribute to ∼90% of the adder-tree resources. Otherwise, the decay in accuracy becomes too severe. Equation (3.8) characterizes the resource usage of the adder-tree in which the first k stages are implemented using 2-bit adders shown in Figure 3.3(d) (including the stage one, which uses the primary exact mode).

$$\overbrace{\sum_{i=1}^{k} \frac{d_{iv}}{3}\frac{1}{2^{i-1}}}^{\text{the first } k \text{ stages}} + \overbrace{\sum_{i=k+1}^{\log d_{iv}} \frac{d_{iv}}{3}\frac{i+1-k}{2^{i-1}}}^{\text{subsequent stages}} \simeq \frac{d_{iv}}{3}(2+\frac{4}{2^k}) \text{ LUT-6} \tag{3.8}$$

We can see that for $k = 1$ – i.e. when none of intermediate stages are truncated – the equation returns $\frac{4}{3}d_{iv}$ which is equal to resources of an exact adder-tree. Setting k to 2, 3, and 4 achieves 25%, 37.5%, and 43.75% resource saving, respectively.

3.3.2 SHEAR*er* Architecture

Recall from Figure 3.1, that the HD encoding procedure needs to convert all input features to equivalent level hypervectors, bind them with the associated ID hypervector, and bundle together (e.g. sum) the resulting hypervectors to generate the final encoding. FPGAs, however, contain limited logic resources as well as on-chip SRAM-based memory blocks (a.k.a BRAMs) to provide high performance with affordable power. Previous work, therefore, break down this step into multiple cycles whereby at each cycle they process d_{seg} dimensions [102, 101, 108]. When processing dimensions $n \cdot d_{seg}$ to $(n+1) \cdot d_{seg}$, those architectures fetch the same dimensions of all $\mathcal{L}$ level hypervectors. Each of d_{seg} adder-trees are augmented with $\mathcal{L}$-to-1 multiplexers in all of their d_{iv} input ports, where the $k^{\text{th}} \leq d_{seg}$ adder-tree's multiplexers are connected to k^{th} dimension of the fetched level hypervectors, and the (quantized) value of associated feature selects the

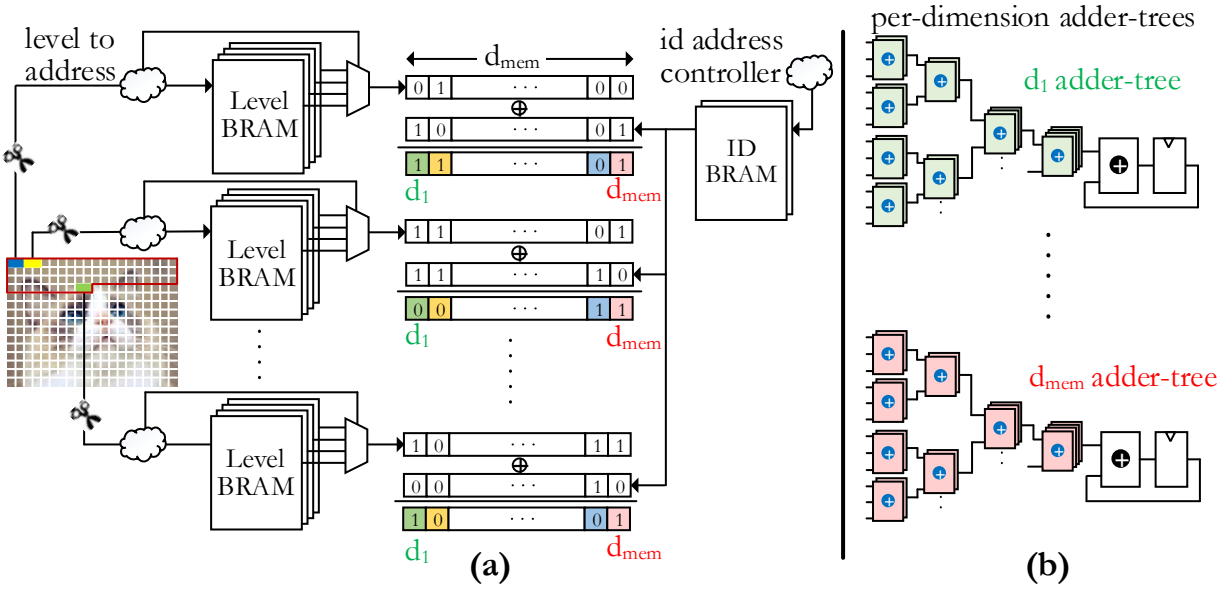


Figure 3.4: SHEAR*er* datapath overview.

right level dimension to pass. The advantage of such architectures is that only $d_{seg} \cdot \mathcal{L}$ bits need to be fetched at each cycle. However, it requires $d_{iv} \cdot d_{seq}$ multiplexers. For a modest $\mathcal{L} = 16$, which translates to 16-input multiplexers occupying four LUTs, the total number of LUTs used for multiplexers will be $4 \cdot d_{iv} \cdot d_{seq}$, the (exact) adder-trees occupy $d_{seq} \cdot \frac{4}{3} d_{iv}$ (in Equation (3.4) we showed that a d_{iv} input exact adder-tree uses $\frac{4}{3} d_{iv}$ LUTs). This means that the augmented multiplexers occupy $3\times$ LUTs of the adder area.In our approximate encoding, this ratio would be even larger as we trim the exact adder. Thus, multiplexer-based implementation overshadows the gain of approximating the adders as we need to preserve the copious multiplexers.

To address this issue, we propose a novel FPGA implementation that relies on on-chip memories rather than adding extra resources. Figure 3.4 illustrates an overview of the SHEAR*er* FPGA architecture. At each cycle, we partially process $\mathcal{F}$ (out of d_{iv}) input features, where $\mathcal{F} \leq d_{iv}$. Our implementation is BRAM-oriented, so each (quantized) feature translates to the address from which the corresponding level hypervector can be read. This entails a dedicated memory block group for each of $\mathcal{F}$ features currently being processed. The number of BRAMs in

a group is equal to group size $= \frac{\mathcal{L} \cdot d_{hv}}{\mathcal{C}_{\text{bram}}}$ as there are $\mathcal{L}$ different level hypervectors of length d_{hv} bits, for a memory capacity of $\mathcal{C}_{\text{bram}}$ bits. Therefore, the number of features $\mathcal{F}$ that can be partially processed in a cycle is limited to $\mathcal{F} < 2\frac{\text{total BRAMs}}{\text{group size}}$. The coefficient 2 is because the BRAMs have two ports from which we can independently read (that is why in Figure 3.4 two pixels share the same BRAM group). The address translator – "level to address" in Figure 3.4) – activates only the right BRAM and row of the group, so the other BRAMs do not dissipate dynamic power. Depending on its configuration, each memory block can deliver up to d_{mem} bits, as indicated in the figure. Certainly, we could double the d_{mem} by duplicating the size of memory groups to process more dimensions per cycle, but then $\mathcal{F}$ – the number of features that can be processed – halves.

Each of d_{mem} fetched level hypervector bit is `XOR`ed with the corresponding bit of the ID (position) hypervector. As detailed in Section 3.2.1, each feature *index* is associated with an ID hypervector, which is a randomly chosen (but fixed) hypervector of length d_{hv}. We thus require $\frac{d_{hv} \cdot d_{iv}}{\mathcal{C}_{\text{bram}}}$ additional BRAM blocks to store ID hypervectors. This further limits the number of features that can be processed in a cycle due to BRAM shortage. To resolve this, we only store a single ID hypervector (seed ID) and generate the other ones by rotating the seed ID, i.e., ID of index k can be obtained by rotating the ID of index 1 (seed ID) by $k-1$. This does not affect the HD accuracy as the resulting ID hypervectors are still *iid* and approximately orthogonal. For the first feature, we need to read d_{mem} bits, while for the subsequent $\mathcal{F}-1$ features we need one more bit as each ID has $d_{mem}-1$ common bits with its predecessor. Therefore we need a data-width of $d_{mem}+\mathcal{F}-1$ for ID memory, meaning that we need $1+\frac{\mathcal{F}}{d_{mem}}$ memory blocks of the seed ID hypervector. Thus, although the seed ID fits in a single BRAM, the required data-width demands more memory blocks. However, this is still significantly smaller than the case of storing all different IDs in BRAM blocks, which either releases BRAMs for processing the features, or power gates the unused BRAMs. Moreover, using seed ID BRAM also saves dynamic power as $d_{mem}+\mathcal{F}-1$ bits are read (compared to $d_{mem} \times \mathcal{F}$ of storing different IDs). It is also noteworthy

that at each cycle the first d_{mem} bits read from the ID memory are passed to the first feature of the features currently being processed (i.e., feature 1, $\mathcal{F}$ + 1, $2\mathcal{F}$ + 1, $\cdots$). Similarly, bits 2 to $d_{mem}+1$ of the fetched ID are passed to the second feature, and so on. Thus, the output of ID BRAMs to processing logic needs a fixed routing.

After `XOR`ing the fetched level hypervectors with the ID hypervectors, each of the d_{mem} approximate adder-trees add up $\mathcal{F}$ binary bits, so the input size of all adders is $\mathcal{F}$. Since the result is only the sum of the first $\mathcal{F}$ features, SHEAR*er* utilizes a buffer to store these partial sums. In the next cycle, the procedure repeats for the next group of features, i.e., features $\mathcal{F}$ + 1 to $2\mathcal{F}$. Therefore, SHEAR*er* produces d_{mem} encoding dimensions in $\frac{d_{iv}}{\mathcal{F}}$ cycles, hence the entire encoding hypervector is generated in $\left\lceil \frac{d_{hv}}{d_{mem}} \right\rceil \times \left\lceil \frac{d_{iv}}{\mathcal{F}} \right\rceil$ cycles.

To make these tangible, in the Xilinx FPGAs we use for experiments, d_{mem} is 64 and $\mathcal{C}_{bram} = 512_{row} \times 64_{col}$. We also noticed that 16 level hypervectors gives the same accuracy of having more, so we set $\mathcal{L} = 16$. We also select the hypervector lengths to be a multiple of 512. Taking the previously mentioned language recognition benchmark [106] as an example, we observed that $d_{hv} = 2{,}560$ provides acceptable accuracy (see Section 3.4 for more details). For this benchmark we thus need group size of $\lceil \frac{16\times2560}{512\times64} \rceil = 2$ BRAMs, where each group can cover two input features. The FPGA we use has a total 445 BRAMs, which can make at most $\lfloor \frac{445}{2} \rfloor = 222$ groups, capable of processing 444 features per cycle. Therefore, we divide 617 input features of the benchmark into two repeating cycles using 310 BRAMs (155 BRAM groups) to process the first 310 features in the first cycle, and the rest 307 cycles in the second cycle, generating $d_{mem} = 64$ encoding dimensions per 2 cycles. All 64 adder-trees have a 1-bit input sizes of 310. The entire encoding takes 2560 dim $\times \frac{2 \text{ cycles}}{64 \text{ dim}} = 80$ cycles. Note that reading from on-chip BRAMs has just one cycle latency and the off-chip memory latency is buried in the computation pipeline.

3.3.3 Software Layer

Because of approximation, the output of encoding and hence the class hypervectors are different than training with exact encoding. Therefore we also need to train the model using the same approximate encoding(s), as the associative search only looks for the *similarity* (rather than exactness) of an approximately encoded hypervector with trained class hypervectors – which are made up by bundling a manifold of encoding hypervectors. Our FPGA implementation is tailored for inference, so we carry out the training step on CPU. We developed an efficient SIMD vectorized Python implementation to emulate the exact and the proposed encoding techniques in software. The emulation of the proposed techniques is straightforward. For instance, for the local majority approximation (Figure 3.3(a)), instead of adding up all d_{iv} hypervectors, we divide them to groups of six hypervectors, add up all six hypervectors of each group, and compare if each resultant dimension is larger than 3. We also break the ties in software by generating a constant vector dictating how the ties of each dimension should be served. This acts as the `MAJ` LUTs of the first stage. Thereafter, we simply add up all these temporary hypervectors to realize the subsequent exact adders. This guarantees to match the software output with approximate hardware's, while we also achieve a fast implementation by avoiding unnecessary imitation of hardware implementation.

In addition to d_{iv} that is the dataset's attribute, d_{hv}, α, `epochs` (number of training epochs) are the other variables of our software implementation. α is the learning rate of HD. As explained in Section 3.1, HD bundles all encoding hypervectors belonging to the same-label data to create the initial class hypervectors. In the subsequent `epochs` iterations, HD updates the class hypervectors by observing if the model correctly predicts the training data. If the model mispredicts an encoded query $\mathcal{H}^l$ of label l as class $\mathcal{C}^{l'}$, HD updates as shown by Equation (3.9). If learning rate α is not provided, SHEAR*er* finds the best α through bisectioning for a certain number of iterations.

$$\mathcal{C}^l = \mathcal{C}^l + \alpha \cdot \mathcal{H}^l \qquad\qquad \mathcal{C}^{l'} = \mathcal{C}^{l'} - \alpha \cdot \mathcal{H}^l \tag{3.9}$$

We supply the software implementation of SHEAR*er* with the number of BRAM and LUT resources of the target FPGA to estimate the architectural parameters according to Section 3.3.2 as well as using the resource utilization formulated in Section 3.3.1. We have also implemented the exact and approximate adder-trees of different input sizes and interpolated their measured power consumption – which is linear w.r.t. the adder size – for different average activities of the adders' primary inputs. Therefore, we calculate the average signal activity observed by the adders according to the values of temporary-generated binding hypervectors (level `XOR` ID). We similarly estimate the toggle rate of BRAMs according to consecutive d_{mem} bits read from BRAMs. As alluded earlier, we do not replicate the hardware implementation in software; we just need to determine each fetched level hypervector belongs to which BRAM group (based on the index of feature), so we can keep track of toggle rates. Using the signal information with an offline look-up table created for activity-power, along with the instantiated resource information calculated as mentioned, during training, SHEAR*er* estimates the power consumption of an application targeted for a specific device.

3.4 Experimental Results

(1) General Setup. We have implemented the SHEAR*er* architecture using Vivado High-Level Synthesis Design Suite on Xilinx Kintex-7 FPGA KC705 Evaluation Kit which embraces a XC7K325T device with 203,800 LUT-6 and 445 36 Kb BRAM memory blocks that we use in 512×64bit configuration. By pipelining the adder-tree stages we could achieve a clock frequency of 200 MHz. We compare the performance and energy results with the high-end NVIDIA GeForce GTX 1080 Ti GPU, and Raspberry Pi 3 embedded processor. We optimize the CUDA implementation by packing the hypervectors within 32-bit integers, so a single logical `XOR` operation can bind 32 dimensions. We use `speech` [106], `activity` [109], and hand-written `digit` [110] recognition as well as a `face` detection dataset [86] as our benchmarks.

Table 3.1: Baseline implementation results.

Parameter ↓ Benchmark →	speech	activity	face	digit
Input features (d_{iv})	617	561	608	784
Hypervector length (d_{hv})	2,560	3,072	6,144	2,048
Baseline accuracy	93.18%	93.91%	95.47%	89.07%

Table 3.2: LUT count for a 512-input adder-tree.

	exact	MAJ	MAJ-2	over-feed	truncate
Synthesis	638	183	116	383	340
Analysis	675	195	116	405	343
Error	5.8%	6.6%	0.0%	5.7%	0.9%

Table 3.1 summarizes the length of hypervectors and associated accuracy of each dataset in the baseline exact mode. For a fair comparison, we first obtained the accuracies using $d_{hv} = 10{,}000$, then decreased it until the accuracies remain within 0.5% of the original values. This avoids over-saturated hypervectors and accuracy drop due to approximation manifests better.

(2) Resource Utilization. To validate the efficiency of the proposed approximation techniques, in addition to holistic high-level performance and energy comparisons, we examine them by synthesizing a 512-input adder-tree. Table 3.2 reports the LUT utilization of the adder implemented in exact and approximate modes. MAJ, MAJ-2, over-feed and truncate refer to the designs of Figure 3.3(a)-(d). It can be seen that our equations in Section 3.3.1 have a modest average error of 3.8%. Especially, it over-estimates the LUT count of both exact and approximate adders, so the resource *saving* estimations remain similar to our predicted values. For instance, synthesis results indicate MAJ (MAJ-2) saves 71.3% (81.8%) LUTs, which is very close to the predicted 71.1% (82.8%).

(3) Accuracy. Table 3.3 summarizes the accuracies of the proposed encodings relative to the exact encoding. LUT saving, which is dataset-independent, is represented again for the comparison purpose. "trunc-3" and "trunc-4" stand for truncated encoding (Figure 3.3(d)) where, respectively, three and four intermediate stages are truncated. Overall, MAJ encoding (one-stage local majority shown in Figure 3.3(a)) achieves an acceptable accuracy with significant resource

Table 3.3: Relative accuracies SHEAR*er* approximate encodings.

	exact	MAJ	MAJ-2	over-feed	trunc-3	trunc-4
speech	93.2%	−0.7%	−2.3%	−0.8%	−0.9%	−1.9%
activity	93.9%	−0.8%	−1.2%	−1.3%	−1.1%	−1.0%
face	95.5%	−1.8%	−3.3%	−1.7%	−1.6%	−1.9%
digit	89.1%	−0.8%	−0.3%	−1.7%	0.1%	−0.1%
average		−1.0%	−1.8%	−1.4%	−0.9%	−1.2%
LUT saving	0	71.1%	82.8%	40.0%	37.5%	43.8%

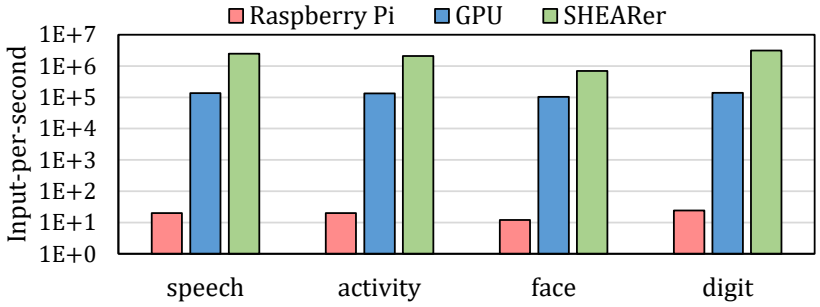


Figure 3.5: Throughput of SHEAR*er* versus Raspberry Pi 3 and Nvidia GTX 1080 Ti. Y-axis is logarithmic scale.

saving, though it is not always the highest-accurate one. For instance, in the `face` detection benchmark, the over-feed and 3-stage truncated encodings offer slightly better accuracy. More interestingly, in the `digit` recognition dataset, trunc-4 shows a negligible -0.1% accuracy drop while trunc-3 even improves the accuracy by 0.1%. This can stem from the fact that emulating the hardware approximation in SHEAR*er*'s software layer takes a long time for the `digit` dataset, so we limited the software to try five different learning rate (α) and repeat the entire training for five times (with `epochs` $= 50$) so the result might be slightly skewed. For the other datasets we conducted the training for 25 times each with 50 `epochs` to average out the variance of results.

(4) Performance. Figure 3.5 compares the throughput of SHEAR*er* FPGA implementation with Raspberry Pi and Nvidia GPU. SHEAR*er* implementation is BRAM-bound, so all the exact and approximate implementations yield the same performance. In Section 3.3.2 we elaborated that the `speech` dataset requires two cycles per $d_{mem} = 64$ dimensions. We can similarly show that `activity` and `digit` datasets also need two cycles per 64 dimensions, while

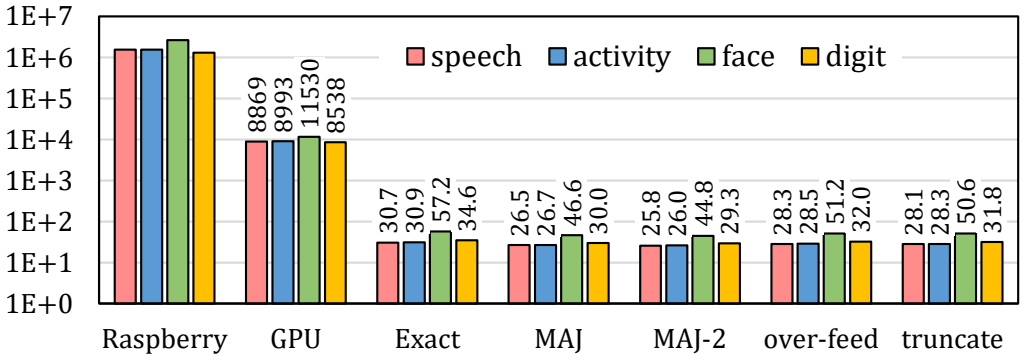


Figure 3.6: Energy (Joule) consumption of SHEAR*er*, Raspberry Pi and GPU for 10 million inference. Y-axis is logarithmic.

`digit` requires three cycles as its level hypervectors are larger ($d_{hv} = 6{,}144$) and occupy more BRAMs. In the worst scenario, SHEAR*er* improves the throughput by 58,333× and 6.7× compared to Raspberry Pi and GPU implementation. On average SHEAR*er* provides a throughput of 104,904× and 15.7× as compared to Raspberry Pi and GPU, respectively. The substantial improvements arise from that SHEAR*er* adds up $\frac{d_{iv}}{2} \times 64$ (e.g., ∼25,000) numbers per cycle while also performs the binding (`XOR` operations) on the fly. However, Raspberry Pi executes sequentially and also its cache cannot fit all the class hypervectors with non-binary dimensions. Note that we assume that dataset is available in the *off-chip* memory (DRAM) of the FPGA. Otherwise, although per-sample latency would be affected, throughput remains the same as the off-chip memory latency is buried in the computation cycles.

(5) Energy Consumption. Figure 3.6 compares the energy consumption of the exact and approximate SHEAR*er* implementations with Raspberry Pi and GPU. We have scaled the energy to 10 million inferences for the sake of illustration (Y-axis is logarithmic). We used Hioki 3334 power meter and NVIDIA system management interface to measure the power consumption of Raspberry Pi and GPU, respectively. We used Xilinx Power Estimator (XPE) to estimate the FPGA power consumption. The average power of Raspberry Pi for all datasets hovers around 3.10 Watt, while this is ∼120 Watt for the GPU. In FPGA implementation, powers showed more variation as the number of active LUTs and BRAMs differ between applications. E.g., The

face dataset with two-stage majority encoding (MAJ-2) consumes 3.11 Watt, while the digit recognition dataset in the exact mode consumes 10.80 Watt. The smaller power consumption of face is mainly because of smaller off-chip data transfer as face has the largest hypervector length and takes 288 cycles to process an entire input, while for digit it takes 64 cycles. On average, SHEAR*er*'s *exact* encoding decreases the energy consumption of by 45,988× and 247× (average of all datasets) as compared to Raspberry Pi and GPU implementations. MAJ-2 encoding of SHEAR*er* consumes the minimum energy, which throttles the energy consumption by 56,044× and 301× compared to Raspberry Pi and GPU, respectively. Note that power improvement of the approximate encodings is not proportional to their resource (LUT) utilization as BRAM power remains the same for all encodings.

3.5 Conclusion

In this chapter, we leveraged the intrinsic error tolerance of HD computing to develop different approximate encodings with varied accuracy and resource utilization attributes. With a modest 1.0% accuracy drop, our approximate encoding reduces the LUT utilization by 71.1%. By effectively utilizing the on-chip BRAMs of FPGA, we also proposed a highly efficient implementation that outperforms an optimized GPU implementation over 15× and surpasses Raspberry Pi by over five orders of magnitude. Our FPGA implementation also consumes a moderate power: a minimum of 3.11 Watt for a face detection dataset using approximate encoding and a maximum of 10.8 Watt for a digit recognition dataset when using exact encoding. Eventually, our implementation reduces the energy consumption by 247× (45,988×) compared to GPU and Raspberry Pi in exact encoding, which further improves by a factor of 1.22× using approximate encoding.

As a proof of concept, we implemented our architectures on a mid-range Kintex-7 device, while given the obtained LUT usage per dimension, the proposed architectures can be

implemented on very low-end FPGA devices such as Microsemi's IGLOO2 series with a few thousands of LUTs and $\sim$10 mW power consumption [111]. While this amount of resource and power consumption might be acceptable for a variety of devices such as gadgets and smartphone, there are many IoT applications and devices that have much stricter resource (area) and energy constraints such as wearable devices (e.g., an eye glass or earbud with integrated seizure detection logic) or remote sensors. Therefore, in the next chapter, we propose custom ASIC implementation for HDC to further improve its efficiency while still offering flexibility in supporting different applications.

Chapter 3, in part, is a reprint of the material as it appears in "SHEARer: Highly-Efficient Hyperdimensional Computing by Software-Hardware Enabled Multifold Approximation" by Behnam Khaleghi, Sahand Salamat, Anthony Thomas, Fatemeh Asgarinejad, Yeseong Kim, and Tajana Rosing which appears in Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Efficient Learning Engine on Edge using Hyperdimensional Computing

In Chapter 2, we proposed techniques to reduce the energy consumption of FPGA-based designs and make them plausible for a wider range of edge applications. In Chapter 3, to cope with the memory and compute overhead of machine learning techniques, we introduced HD computing and proposed exact and approximate HDC implementation that allows implementing on low-end FPGAs. However, although low-end FPGA devices are used in certain IoT use cases, their large form-factor and constant leakage power consumption hinders their utilization in extremely low-power use cases such as wearable devices and other tiny sensors. In this chapter, we show conventional processing systems are incapable of taking full advantage of the highly-parallel bit-level operations of HDC, which makes a custom ASIC design of HDC indispensable. However, existing HDC encoding techniques do not cover a broad range of applications to make a fixed design plausible. Therefore, in this chapter, we first propose a novel encoding that achieves high accuracy for diverse applications. Then, we leverage the proposed encoding and design a highly efficient and flexible ASIC accelerator, suited for low-power edge domain, and capable of

running classification (both learning and inference) as well as clustering.

4.1 Introduction

As explained in Chapter 3, HDC uses specific algorithms to encode raw inputs to a high-dimensional representation of hypervectors with $\mathcal{D}_{hv} \approx 2-5K$ dimensions. The encoding takes place by deterministically associating each element of an input with a binary or bipolar (± 1) hypervector and bundling (element-wise addition) the hypervectors of all elements to create the encoded hypervector. Training is straightforward and involves bundling all encoded hypervectors of the same category. For inference, the query input is encoded to a hypervector in the same fashion and compared with all class hypervectors using a simple similarity metric such as cosine.

The bit-level massively parallel operations of HDC do not accord well with conventional CPUs/GPUs due to, e.g., memory latency and data movement of large vectors and the fact that these devices are over-provisioned for majorly binary operations of HDC. Previous works on custom HDC accelerators support a limited range of applications or achieve low accuracy. The authors of [36] and [37] propose custom HDC inference designs that are limited to a specific application. More flexible HDC inference ASICs are proposed in [38] and [39], but as we quantify in Section 4.3.2, the utilized encoding techniques achieve poor accuracy for particular applications such as time-series. The authors of [40] propose a trainable HDC accelerator, which yields 9% lower accuracy than baseline ML algorithms. An HDC-tailored processor is proposed in [112], but it consumes $\sim 1-2$ orders of magnitude more energy than ASIC counterparts. The in-memory HDC platform of [113] uses low-leakage PCM cells to store hypervectors, but its CMOS peripherals throttle the overall efficiency.

In this chapter, we propose GENERIC (highly efficient learning engine on edge using hyperdimensional computing) for highly efficient and accurate trainable classification and clustering. Our primary goal is to make GENERIC compact and low-power to meet year-long

battery-powered operation, yet fast enough during training and burst inference, e.g., when it serves as an IoT gateway. To this end, we make the following contributions.

(1) We propose a novel HDC encoding that yields high accuracy in various benchmarks. Such a generic encoding is fundamental to develop a custom yet flexible circuit.

(2) We perform a detailed comparison of HDC and various ML techniques on conventional devices and point out the failure of these devices in unleashing HDC advantages.

(3) We propose the GENERIC flexible architecture that implements accurate HDC-based trainable classification and clustering.

(4) GENERIC benefits from extreme energy reduction techniques such as application-opportunistic power gating, on-demand dimension reduction, and error-resilient voltage over-scaling.

(5) Comparison of GENERIC with the state-of-the-art HDC implementations reveals GENERIC improves the classification accuracy by 3.5% over previous HDC techniques and 6.5% over ML techniques. GENERIC improves the inference energy consumption by 528×, and training energy by 1590× over the most-efficient ML algorithm, while achieving higher accuracy. Compared to previous HDC accelerators [38] and [40], GENERIC reduces the energy consumption by 4.1× and 15.7×, respectively.

4.2 Hyperdimensional Computing Background

In Chapter 3, we introduced the concepts of HDC learning and inference. Therefore, in this section we only introduce different encoding techniques of HDC before evaluating them. We also explain HDC-based clustering we the proposed accelerator is capable of clustering using HDC.

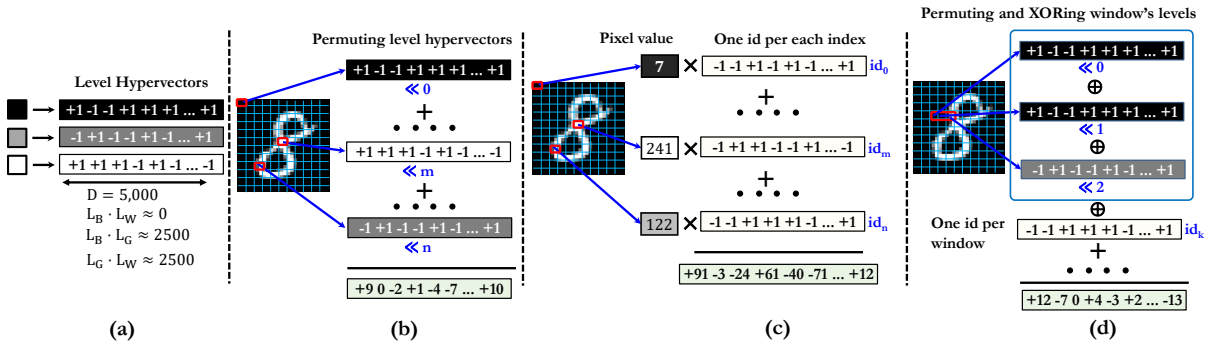


Figure 4.1: (a) Level hypervectors, (b) permutation encoding, (c) random projection encoding, (d) proposed GENERIC encoding.

4.2.1 Encoding

In Chapter 3 we introduced base-level (aka id-level or id-vector) encoding. There are several other encoding techniques to map the inputs to high-dimensional space for subsequent learning tasks, such as permutation and random projection encoding shown in Figure 4.1. Each encoding technique preserves a particular distance (such as L1 or angular distance) suitable for particular data types. We denote any input in the low-dimensional space as a vector $\mathcal{X} = (x_1, \cdots x_{d_{iv}})$ where $x_i \in \mathbb{R}$ and d_{iv} is the number of features (dimensionality of the input vector in the original space).

Base-level is discussed in the previous chapter. To recap, it quantizes the input features to Q levels and assigns a binary or bipolar (± 1) hypervector of length $\mathcal{D}_{hv}$. We explained the process of generating the level vectors in Chapter 3. In addition to level hypervectors, a random base $\vec{\mathcal{B}}_i$ (aka *id*) hypervector per each feature position is also used to account for the order of features in an input [114]. Equation (4.1) formulates the base-level encoding.

$$\vec{\mathcal{H}}(\mathcal{X}) = \sum_{i=1}^{d_{iv}} \vec{\mathcal{B}}_i \cdot \vec{\mathcal{L}}(x_i) \tag{4.1}$$

Permutation encoding uses the same concept of level hypervectors as in base-level encoding but accounts for positional information using circular shift instead of binding with

base hypervectors. In Equation (4.2), $\rho^k(X)$ indicates circular shift of X by k indexes, which has replaced binding by $\vec{\mathcal{B}}_k$ of Equation (4.1). Therefore, after selecting the proper level hypervector of feature x_i, it is shifted by $i-1$ indexes and is added up with the same of the rest of x_is.

$$\vec{\mathcal{H}}(\mathcal{X}) = \sum_{i=1}^{d_{iv}} \rho^{(i-1)}\left(\vec{\mathcal{L}}(x_i)\right) \tag{4.2}$$

Random Projection (RP) encoding, similar to base-level, uses base hypervectors to take care of positional information, but instead of level hypervectors directly uses the scalar values of features ('x_i's) as shown by Equation (4.3a), in which $x_i \times \vec{\mathcal{B}}_i$ is a scalar-vector multiplication between a single feature x_i and the base hypervector of position i ($\vec{\mathcal{B}}_i$). RP encoding may be accompanied by a quantization scheme such as the sign function to restrict the embedding to $\mathcal{H}$.

$$\vec{\mathcal{H}}(\mathcal{X}) = \sum_{i=1}^{d_{iv}} x_i \times \vec{\mathcal{B}}_i \tag{4.3a}$$

$$\vec{\mathcal{H}}(\mathcal{X}) = \text{sign}\Big(\sum_{i=1}^{d_{iv}} x_i \times \vec{\mathcal{B}}_i\Big) \tag{4.3b}$$

Note that we can rearrange all d_{iv} base hypervectors into a $\mathcal{D}_{hv} \times d_{iv}$ matrix. Therefore, Equation (4.3) can be rewritten as $\vec{\mathcal{H}}(\mathcal{X}) = \mathcal{B}' \times \mathcal{X}$, a matrix-vector multiplication, in which each column i of $\mathcal{B}'$ is base hypervector $\vec{\mathcal{B}}_i$.

N-gram encoding is used in applications that the global position of features does not provide meaningful information, e.g., the relative position of words when detecting a language. N-gram encoding applies circular shift over the level hypervectors of N consecutive features within a sliding window (zero shift for the first level hypervector and $N-1$ shift for the N^{th} one), obtains the dot-product/XOR of these permuted hypervectors, and accumulates/bundles the result of all $d_{iv}-N+1$ windows [36]. Typically $N \in \{2,3,4\}$. Note that N-gram encoding is naturally incapable of accounting for global positional information and is expected to yield low accuracy

Algorithm 3: HDC Clustering Algorithm

Input: $\mathcal{S}_{\mathcal{X}}$: list of all inputs $\mathcal{X}$ in low-dimensional space
Input: k: number of clusters, T number of epochs
Output: $\mathcal{C}_{\mathcal{X}}$: cluster of all inputs $\mathcal{X}$

for $i \in 1 : |\mathcal{S}_{\mathcal{X}}|$ **do**
  $\mathcal{H}^i = \text{encode}(\mathcal{X}^i)$
for $i \in 1 : k$ **do**
  $\mathcal{C}^i_{prev} = \mathcal{H}^{\text{random}(1,|\mathcal{S}_{\mathcal{X}}|)}$ // initialize centroids
for $t \in 1 : T$ **do**
  $\mathcal{C}_{next} = \emptyset$
  for $i \in 1 : |\mathcal{S}_{\mathcal{X}}|$ **do**
    $c^\star = \text{argmax}\ \delta(\mathcal{H}^i, \mathcal{C}^j_{prev})\ \forall\ j \in [1,k]$
    $\mathcal{C}^{c^\star}_{next} \mathrel{+}= \mathcal{H}^i$
    $\mathcal{C}_{\mathcal{X}}(\mathcal{X}^i) = c^\star$ // cluster of the i^{th} input is $c^\star$
  $\mathcal{C}_{prev} = \mathcal{C}_{next}$
return $\mathcal{C}_{\mathcal{X}}$

in many application domains.

$$\vec{\mathcal{H}}(\mathcal{X}) = \sum_{i=1}^{d_{iv}-N} \prod_{j=0}^{N-1} \rho^{(j)} \left(\vec{\mathcal{L}}(x_{i+j}) \right) \tag{4.4}$$

4.2.2 HDC Clustering

The similarity of hypervectors indicates their proximity [31], which can be used to cluster data in the hyperspace. Algorithm 3 summarizes the HDC clustering, which in essence is similar to the k-means clustering algorithm [114]. Initially, k encoded input hypervectors are randomly chosen to serve as centroids. After that, HDC clustering iterates over the remaining encoded data, and according to the similarity score of each encoded hypervector with the current centroids, it adds up all hypervectors that belong (most similar) to the same centroid to create the centroids of the updated clusters. This procedure repeats for a predefined number of epochs or until convergence, when none of the centroids are updated (the new centroids are the same as previous ones).

4.3 Proposed HDC Encoding

4.3.1 GENERIC Encoding

The encoding techniques discussed in Section 4.2.1 achieve low accuracy for certain datasets such as language identification which generally need extracting local subsequences of consecutive features, without considering the global order of these subsequences (see subsection 4.3.2). Previous studies use *ngram* encoding for such datasets [36, 37, 115]. Ngram encoding extracts all subsequences of length n (usually $n \in \{3-5\}$) in a given input, encodes all these subsequences and aggregates them to produce the encoded hypervector. However, ngram encoding achieves very low accuracy for datasets such as images or voices in which the spatio-temporal information of should be taken into account.

We propose a new encoding, dubbed GENERIC, to cover a more versatile set of applications. As shown in Figure 4.1(d), our encoding processes sliding windows of length n by applying the permutation encoding. That is, for every window consisting of elements $\{x_k, x_{k+1}, x_{k+2}\}$ (for n=3), three level hypervectors are selected, where $\ell(x_k)$, $\ell(x_{k+1})$, and $\ell(x_{k+2})$ are permuted by 0, 1, and 2 indexes, respectively. The permuted hypervectors are XORed element-wise to create the *window hypervector*. The permutation accounts for positional information within a window, e.g., to distinguish "abc" and "bca". To account for *global* order of features, we associate a random but constant *id* hypervector with each window, which is XORed with the window hypervector to perform *binding*. To skip the global binding in certain applications, *id* hypervectors are set to $\{0\}^{\mathcal{D}_{hv}}$. Equation (4.5) formalizes our encoding, where $\rho^{(j)}$ indicates permutation by j indexes, $\prod$ multiplies (XOR in binary) the levels of i^{th} window, id_i applies the binding *id*, and $\sum$ adds up the window hypervector for all windows of d elements.

$$\mathcal{H}(\mathcal{X}) = \sum_{i=1}^{d-n+1} \Big(id_i \cdot \prod_{j=0}^{n-1} \rho^{(j)}\big(\ell(x_{i+j})\big) \Big) \tag{4.5}$$

Table 4.1: Accuracy of HDC and ML algorithms.

Dataset	HDC Algorithms					ML Algorithms			
	RP	level-id	ngram	permute	GENERIC	MLP	SVM	RF	DNN
CARDIO	83.0%	88.1%	88.1%	88.2%	**91.8%**	86.4%	86.4%	95.3%	90.1%
DNA	99.3%	99.3%	99.7%	99.3%	**99.7%**	99.5%	99.5%	99.5%	99.8%
EEG	46.8%	77.5%	83.1%	78.3%	**83.1%**	56.8%	75.4%	80.1%	60.2%
EMG	53.6%	90.9%	90.8%	91.1%	**90.9%**	91.0%	89.2%	83.6%	89.4%
FACE	95.3%	95.0%	73.3%	96.1%	**95.7%**	95.5%	97.3%	92.5%	96.7%
ISOLET	93.2%	93.5%	38.9%	93.5%	**93.1%**	95.0%	96.0%	92.2%	94.4%
LANG	8.2%	75.9%	100.0%	52.8%	**100.0%**	5.4%	30.8%	10.3%	99.9%
MNIST	94.6%	89.4%	53.0%	89.3%	**94.0%**	96.7%	97.9%	96.0%	99.1%
PAGE	96.1%	91.6%	91.7%	91.7%	**91.8%**	96.5%	96.9%	97.4%	95.8%
PAMAP2	83.0%	94.6%	60.9%	95.8%	**93.8%**	92.9%	91.9%	95.6%	96.1%
UCIHAR	93.4%	94.6%	64.9%	94.7%	**94.9%**	94.6%	95.8%	95.6%	96.5%
Mean	77.0%	90.0%	76.8%	88.3%	**93.5%**	82.8%	87.0%	85.3%	92.5%
STDV	27.5%	6.9%	19.2%	12.4%	**4.4%**	26.9%	19.0%	24.4%	10.8%

We use n=3 as it achieved the highest accuracy (on average) for our examined benchmarks (see subsection 4.3.2), however, GENERIC architecture can adjust the value of n for every application.

4.3.2 Accuracy Comparison

We compiled eleven datasets from different domains, consisting of the benchmarks described in [40], seizure detection by skull surface EEG signals, and user activity recognition by motion sensors (PAMAP2) [116]. We implemented the HDC algorithms using an optimized Python implementation that leverages SIMD operations. For ML techniques, we used Python scikit-learn library [117]. We discarded the results of logistic regression and k-nearest neighbors as they achieved lower accuracy. For DNN models of benchmarks, we used AutoKeras library [118] for automated model exploration.

Table 4.1 summarizes the accuracy results (RP: random projection, MLP: multi-layer perceptron, SVM: support vector machine, RF: random forest). The proposed GENERIC encoding achieves 3.5% higher accuracy than the best baseline HDC (level-id), 6.5% higher than best baseline ML (SVM), and 1.0% higher than DNN. The RP encoding fails in time-series datasets

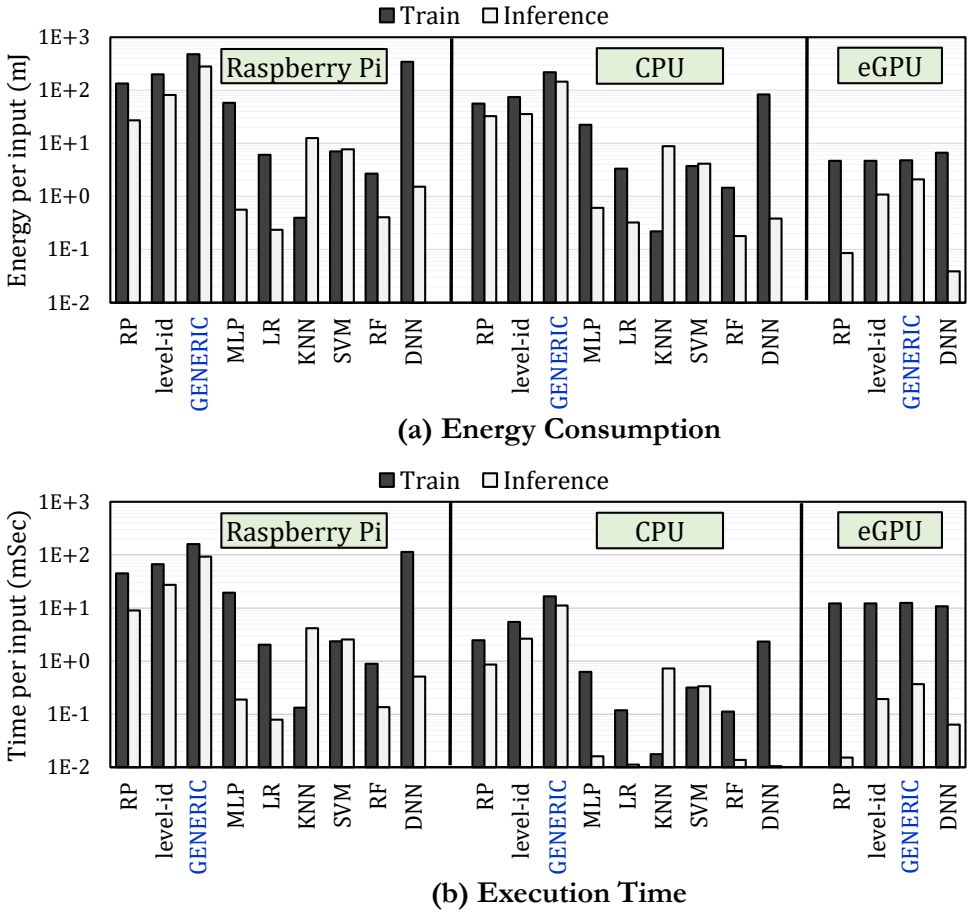


Figure 4.2: (a) Energy consumption and (b) execution time of HDC and ML algorithms on different devices.

that require temporal information (e.g., EEG). As explained in subsection 4.3.1, the ngram encoding [36, 115] do not capture the global relation of the features, so it fails in datasets such as speech (ISOLET) and image recognition (MNIST). Except for the ngram and the proposed GENERIC, other HDC techniques fail in the LANG (text classification) as they enforce capturing sequential information and ignore subsequences.

4.3.3 Efficiency on Conventional Hardware

HDC's operations are simple and highly parallelizable, however, conventional processors are not optimized for binary operations such as one-bit accumulation. Also, the size of hypervec-

tors in most settings becomes larger than the cache size of low-end edge processors, which may impose significant performance overhead. For a detailed comparison, we implemented the HDC and ML algorithms on the datasets of subsection 4.3.2 on a Raspberry Pi 3 embedded processor and NVIDIA Jetson TX2 low-power edge GPU, and also a desktop CPU (Intel Core i7-8700 at 3.2 GHz) with a larger cache. We used Hioki 3334 power meter to measure the power of the Raspberry Pi.

Figure 4.2 compares the training and inference (a) energy consumption and (b) execution time of the algorithms, reported as the geometric mean of all benchmarks (for eGPU, we omitted the results of conventional ML as it performed worse than CPU for a variety of libraries we examined). We can observe that (i) conventional ML algorithms, including DNN, unanimously consume smaller energy than HDC on all devices, (ii) GENERIC encoding, due to processing multiple hypervectors per window, is less efficient than other HDC techniques, and (iii) our eGPU implementation, by data packing (for parallel XOR) and memory reuse, significantly improves the HDC execution time and energy consumption. For instance, eGPU improves the energy usage and execution time of GENERIC inference by 134× and 252× over running on low-end Raspberry Pi (70× and 30× over CPU). However, GENERIC running on eGPU still consumes 12× (3×) more inference (train) energy, with 27× (111×) higher execution time than the most efficient baseline (random forest). Nonetheless, eGPU numbers imply substantial energy and runtime reduction potential for HDC by effectively taking advantage of low-precision operations (achieved by bit-packing in eGPU) and high parallelism.

4.4 GENERIC Architecture

4.4.1 Overview

Figure 4.3 shows the main components of GENERIC architecture. The main inputs include (i) *input* port to read an input (including the *label* in case of training) from the serial

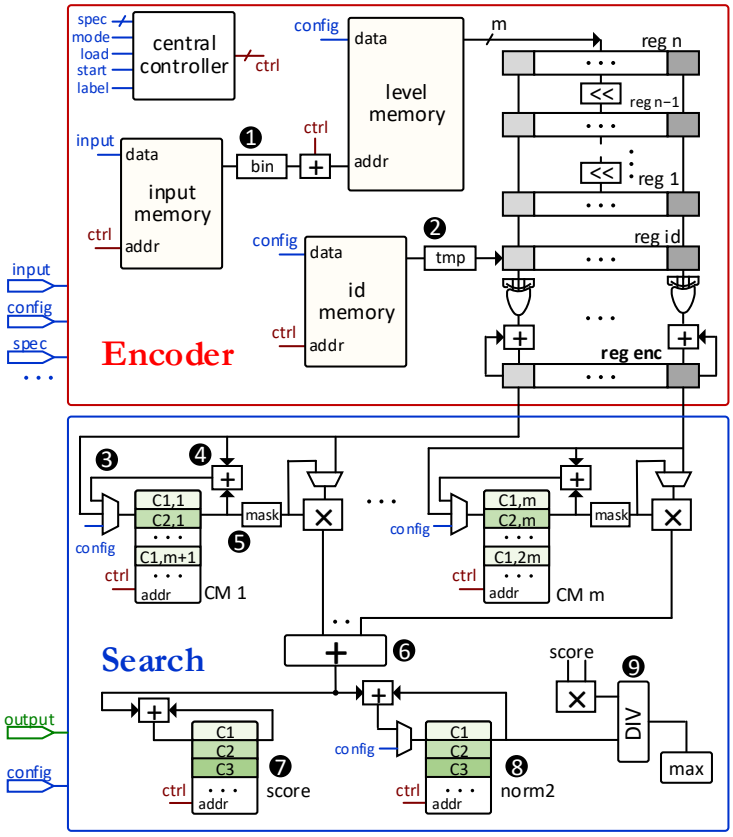


Figure 4.3: Overview of GENERIC architecture.

interface element by element and store in the input memory before starting the encoding, (ii) *config* port to load the level, *id*, and class hypervectors (in case of offline training), and (iii) *spec* port to provide the application characteristics to the controller, such as $\mathcal{D}_{hv}$ dimensionality, d elements per input, n length of window, n_C number of classes or centroids, bw effective bit-width, and *mode* (training, inference, or clustering). *Output* port returns the labels of inference or clustering.

The controller, by using *spec* data, handles the programmability of GENERIC and orchestrates the operations. For instance, the encoder generates m=16 (architectural constant) partial dimensions after each iteration over the stored input, where the variable $\mathcal{D}_{hv}$ signals the end of encoding to finalize the search result, d denotes the number of input memory rows to be proceeded to fetch features (i.e., the exit condition for counter), n_C indicates the number of

class memory rows that need to be read for dot-product and so on. The class memory layout of GENERIC also allows trade off between the hypervectors length D_{hv} and supported classes n_C. By default, GENERIC class memories can store D_{hv}=4K for up to n_C=32 classes. For an application with less than 32 classes, higher number of dimensions can be used (e.g., 8K dimensions for 16 classes). We further discuss it in subsection 4.4.3. These application-specific input parameters enable GENERIC the flexibility to implement various applications without requiring a complex instruction set or reconfigurable logic.

In the following, we explain how GENERIC implement different HDC stages.

4.4.2 Classification and Clustering

Encoding and Inference: Features are fetched one by one from the input memory and quantized to obtain the level bin, and accordingly, m (16) bits of the proper level hypervector are read. The levels are stored as m-bit rows in the level memory. The stacked registers (`reg` n to 1) facilitate storing and on-the-fly sliding of level hypervectors of a window. Each pass over the input features generates m encoding dimensions, which are used for dot-product with the classes. The class hypervectors are distributed into m memories (`CM` 1 to `CM` m) to enable reading m consecutive dimensions at once. The dot-product of partial encoding with each class is summed up in the pipelined adder ❻, and accumulated with the dot-product result of previous/next m dimensions in the score memory ❼.

After $\frac{\mathcal{D}_{hv}}{m}$ iterations, all dimensions are generated, and the dot-product scores are finalized. We use cosine similarity metric between the encoding vector $\mathcal{H}$ and class $\mathcal{C}_i$: $\delta_i = \frac{\mathcal{H} \cdot \mathcal{C}_i}{\|\mathcal{H}\|_2 \times \|\mathcal{C}_i\|_2}$; hence, we need to normalize the dot-product result with L2 norms. The $\|\mathcal{H}\|_2$ can be removed from the denominator as it is a constant and does not affect the rank of classes. In addition, to eliminate the square root of $\|\mathcal{C}_i\|_2$, we modify the metric to $\delta_i = \frac{(\mathcal{H} \cdot \mathcal{C}_i)^2}{\|\mathcal{C}_i\|_2^2}$ without affecting the predictions. The `norm2` memory of Figure 4.3 ❽ stores the *squared* L2 norms of classes, and similarly, the *squared* score is passed to the divider ❾. We use an approximate log-based

division [119].

Training and Retraining: In the first round of training, i.e., model initialization, encoded inputs of the same class/label are accumulated. It is done through the adder ❹ and mux ❸ of all class memories. The controller uses the input label and the iteration counter to activate the proper memory row. In the next retraining epochs, the model is examined and updated in case of misprediction. Thus, during retraining, meanwhile performing inference on the training data, the encoded hypervector is stored in temporary rows of the class memories (through the second input of mux ❸). If updating a class is required, the class rows are read and latched in the adder ❹, followed by reading the corresponding encoded dimensions from the temporary rows and writing the new class dimensions back to the memory. Hence, each update takes $3\times\frac{\mathcal{D}_{hv}}{m}$ cycles. Training also requires calculating the squared L2 norm of classes in the `norm2` memory ❽. As it can be seen in Figure 4.3, the class memories are able to pass the output into both ports of the multipliers (one direct and another through the mux) to calculate and then accumulate the squared elements.

Clustering: GENERIC selects the first k encoded inputs as the initial cluster centroids and initializes k centroids in the class memories. It allocates two sets of memory rows for temporary data; one for the incoming encoding generated in the encoding module and another for the copy centroids (as mentioned in Section 4.2.2, clustering generates a new copy instead of direct update). Similarity checking of the encoding dimensions with the centroids is done pipelined similar to inference, but the encoded dimensions are stored to be added to the copy centroid after finalizing the similarity checking. After finding the most similar centroid, the copy centroid is updated by adding the stored hypervector (similar to retraining). The copy centroids serve as the new centroids in the next epoch.

4.4.3 Energy Reduction

We take advantage of the properties of GENERIC architecture and HDC for utmost energy efficiency. The following elaborates energy-saving techniques that benefit GENERIC. These

techniques can also be applied to other HDC accelerators.

id Memory Compression: The *id* memory naturally needs 1K×4K=512 KB (for up to to 1K features per input, and $\mathcal{D}_{hv}$=4K dimensions) which occupies a large area and consumes huge power. However, GENERIC generates *id*s on-the-fly using a seed *id* vector, where k^{th} *id* is generated by permuting the seed *id* by k indexes. Therefore, the *id* memory shrinks to 4 Kbit, i.e., 1024× reduction. Permutation preserves the orthogonality. It is implemented by the *tmp* register in Figure 4.3❷, by which, for a new window, the `reg id` is right-shifted and one bit of *tmp* is shifted in. The *tmp* register helps to avoid frequent access to the *id* memory by reading m (16) bits at once and feeding in the next m cycles.

Application-opportunistic Power Gating: For an application with n_C classes and using $\mathcal{D}_{hv}$ dimensions, GENERIC stripes the dimensions 1 to m (16) of its 1^{st} class vector in the 1^{st} row of m class memories, the 2^{nd} class vector in the 2^{nd} row, and so on (see Figure 4.3). The next m dimensions of the 1^{st} class vector are therefore written into $n_C + 1^{\text{th}}$ row, followed by the other classes. Thus, GENERIC always uses the first $\frac{n_C \times \mathcal{D}_{hv}}{32 \times 4\text{K}}$ portion of class memories.

The applications of Section 4.3.2, on average, fill 28% of the class memories (minimum 6% for EEG/FACE, and maximum 81% for ISOLET) using $\mathcal{D}_{hv}$=4K dimensions. Accordingly, GENERIC partitions each class memory into four banks and power gates the unused banks. With four banks, 1.6 out of four banks are activated on average, leading to 59% power saving. With more fine-grained eight banks, 2.7 banks (out of eight) become active, saving 66% power. However, eight banks impose 55% area overhead compared to 20% of four banks (see Section 4.5.1 for setup). We concluded that the four-bank configuration yields the minimum area×power cost. Since the power gating is static (permanent) for an application, no wake-up latency or energy is involved.

On-demand Dimension Reduction: GENERIC can trade the energy consumption and performance with accuracy. Recall that GENERIC generates m dimensions of the encoding per iteration over the features. By feeding a new $\mathcal{D}_{hv}$ value as input, GENERIC can seamlessly

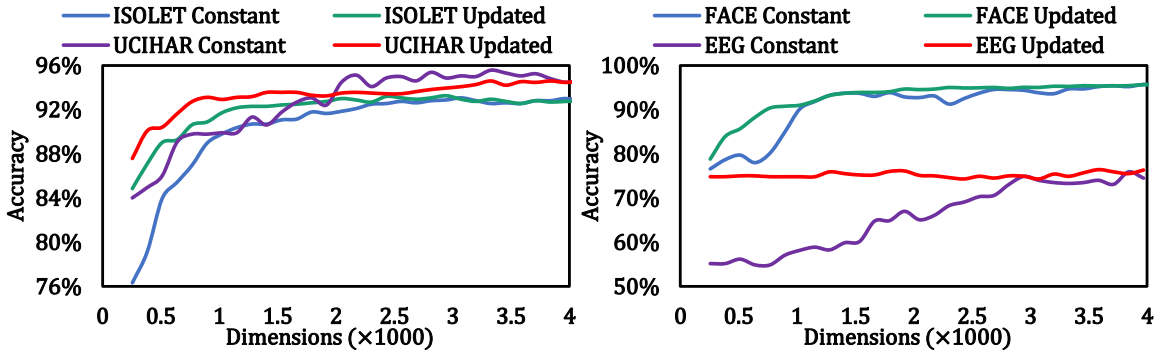


Figure 4.4: Accuracy with constant and updated L2 norm.

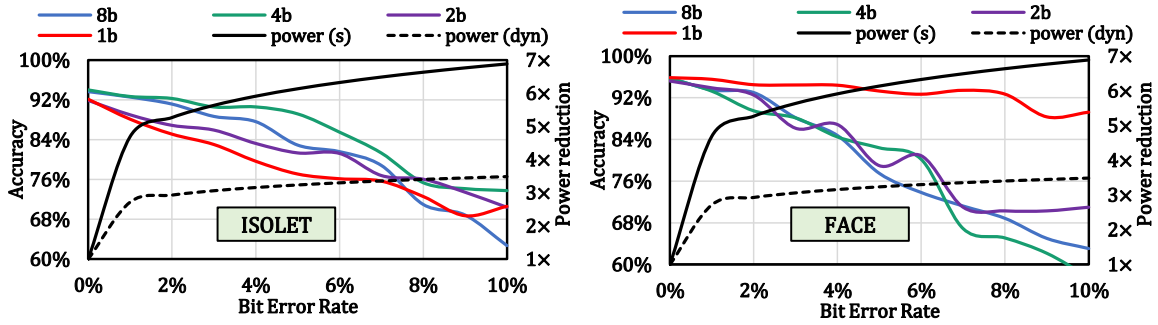


Figure 4.5: Accuracy and power reduction with respect to memory error.

use the new dimension count by updating the counter exit condition, so smaller hypervectors of the encoding and class hypervectors will be used. Nevertheless, GENERIC stores the squared L2 norms of the whole classes for similarity metric ($\delta_i = \frac{(\mathcal{H}\cdot C_i)^2}{\|C_i\|_2^2}$) while for arbitrary reduced encoding dimensions, only the corresponding elements (and their L2 norms) of the classes are needed. As Figure 4.4 shows, using the old (*Constant*) L2 values causes significant accuracy loss compared to using the recomputed (*Updated*) L2 norm of sub-hypervectors. The difference is up to 20.1% for EEG and 8.5% for ISOLET. To address this issue, when calculating the squared L2 norms during the training, GENERIC stores the L2 norms of every 128$^{\text{th}}$-dimension sub-class in a different row of the `norm2` memory ❽.Thus, dimensions can be reduced with a granularity of 128 while keeping the `norm2` memory small (2 KB for 32 classes).

Voltage Over-scaling: GENERIC has to use 16-bit class dimensions to support training. As a result, the large class memories consume ∼80% of the total power. HDC exhibits notable

tolerance to the bit-flip of vectors [80], which can be leveraged to over-scale the memory voltage without performance loss. Figure 4.5 shows the accuracy of select benchmarks (ISOLET and FACE) with respect to the class memory error. The static (*s*) and dynamic (*dyn*) power saving as a result of corresponding voltage scaling (without reducing clock cycle) is also shown in the right axis (based on the measured data of [120]). The figure shows the result of the HDC models with different bit-width (*bw* input parameter of GENERIC) of classes by loading a quantized HDC model (the mask unit ❺ in the architecture masks out the unused bits). As it can be seen, error tolerance not only depends on application but also on the bit-width. 1-bit FACE model shows a high degree of error tolerance (hence, power saving) by up to 7% bit-flip error rate, while ISOLET provides acceptable accuracy by up to 4% bit-flip using a 4-bit model. Quantized elements also reduce the dynamic power of dot-product.

Voltage over-scaling also depends on the application's sensitivity to dimension reduction and its workload. For instance, FACE has a higher tolerance to voltage scaling than dimension reduction (see Figure 4.4). On the other hand, ISOLET is more sensitive to voltage reduction but achieves good accuracy down to 1K dimensions (Figure 4.4), which means 4× energy reduction compared to 4K dimensions. Thus, voltage over-scaling for ISOLET is only preferred in workloads with a higher idle time where the static power dominates (voltage scaling reduces the static power more significantly).

4.5 Results

4.5.1 Setup

We implemented GENERIC at the RTL level in SystemVerilog and verified the functionality in Modelsim. We used Synopsys Design Compiler to synthesize GENERIC targeting 500 MHz clock with 14 nm Standard Cell Library of GlobalFoundries. We used Artisan memory compiler to generate the SRAM memories. The level memory has a total size of $64 \times 4\text{K} = 32\text{KB}$

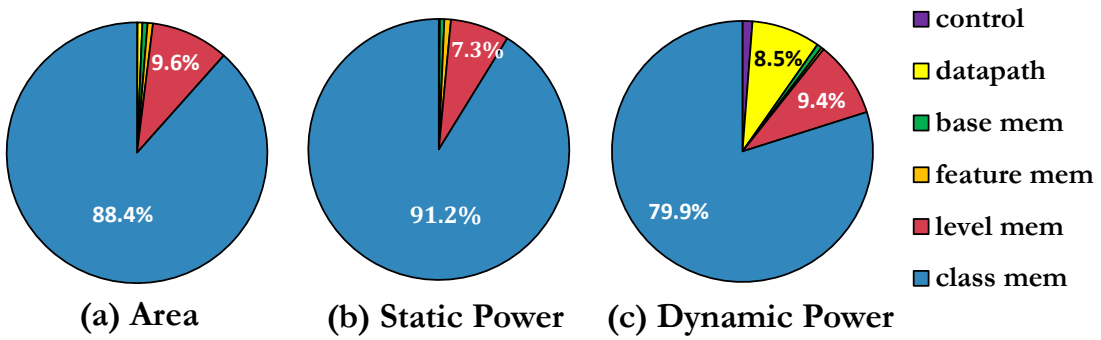


Figure 4.6: Area and power breakdown of GENERIC components.

for 64 bins, the feature memory is 1024×8b, and class memories are 8K×16b (16 KB each). We obtained the power consumption using Synopsys Power Compiler. GENERIC occupies an area of 0.30 mm^2 and consumes a *worst-case* static power of 0.25 mW when all memory banks are active. For datasets of Section 4.3.2, GENERIC consumes a static and dynamic power of 0.09 mW, and 1.79 mW, respectively (without voltage scaling). Figure 4.6 shows the area and power breakdown. Note that the level memory contributes to less than 10% of area and power. Hence, using more levels does not considerably affect the area or power.

4.5.2 Classification Evaluation

Training: Since previous HDC ASICs have not reported training energy and performance, we compare the per-input energy and execution time of GENERIC training with RF (random forest, most efficient baseline) and SVM (most accurate conventional ML) on CPU, and DNN and HDC on eGPU. Figure 4.7 shows the average energy and execution time for the datasets of Section 4.3.2. GENERIC improves the energy consumption by 528× over RF, 1257× over DNN, and 694× over HDC on eGPU (which, as discussed in Section 4.3.3, is the most efficient baseline device for HDC). GENERIC consumes an average 2.06 mW of training power. It also has 11× faster train time than DNN and 3.7× than HDC on eGPU. RF has 12× smaller train time than GENERIC, but as we mentioned, the overall energy consumption of GENERIC is significantly

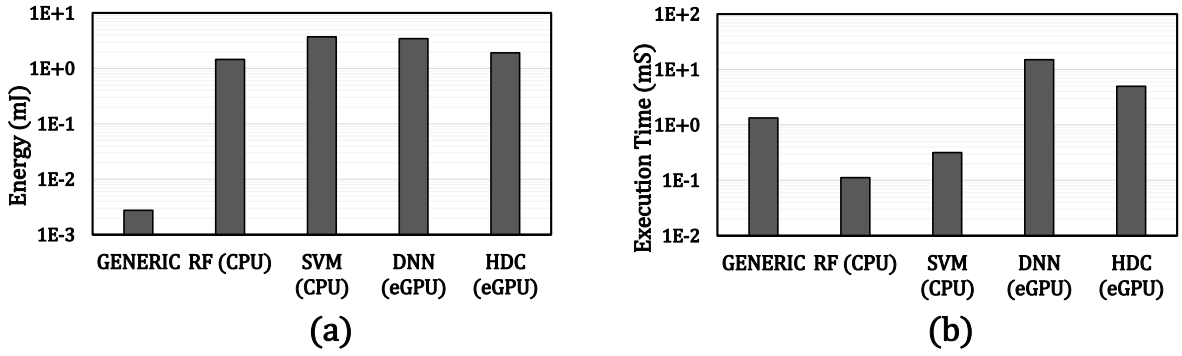


Figure 4.7: (a) Training energy and (b) execution time.

(528×) smaller than RF. Also, we used constant 20 epochs for GENERIC training while the accuracy of most datasets saturates after a few epochs.

Inference: We compare the energy consumption of GENERIC inference with previous HDC platforms from Datta et al. [40], and tiny-HD [38]. We scale their report numbers to 14 nm according to [121] for a fair comparison. We also include the RF (most efficient ML), SVM (most-accurate ML) and DNN on HDC on eGPU (most-efficient HDC baseline). Figure 4.8 compares the energy consumption of GENERIC and aforementioned baselines. Since GENERIC achieves significantly higher accuracy than previous work (e.g., 10.3% over [40]), GENERIC−LP applies the low-power techniques of Section 4.4.3 to leverage this accuracy benefit. GENERIC−LP improves the baseline GENERIC energy by 15.5× through dimension reduction and voltage over-scaling. GENERIC−LP consumes 15.7× and 4.1× less energy compared to [40] and tiny-HD [38], respectively. Note that despite tiny-HD [38], GENERIC supports training which makes it to use larger memories. GENERIC is is 1593× and 8796× more energy-efficient than the most-efficient ML (RF) and eGPU-HDC, respectively.

4.5.3 Clustering Evaluation

Table 4.2 compares the normalized mutual information score of the K-means and HDC for the FCPS [122] benchmarks and the Iris flower dataset. On average, K-means achieves slightly

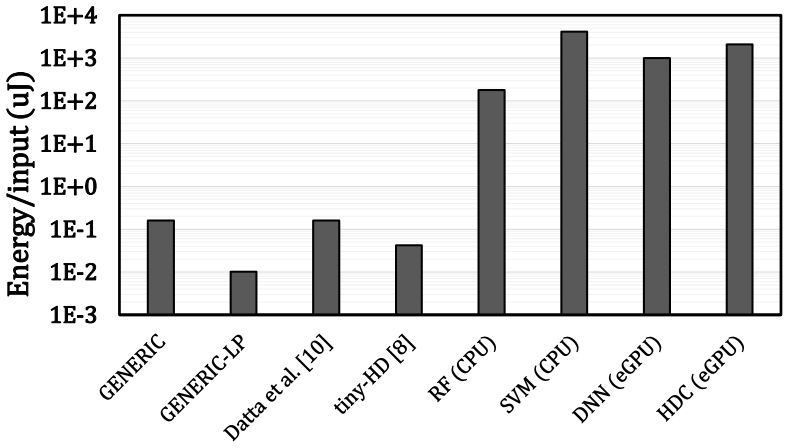


Figure 4.8: Inference energy of GENERIC vs baselines.

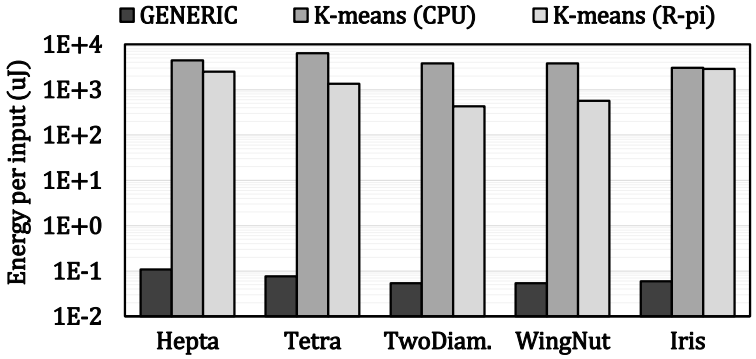


Figure 4.9: GENERIC and K-means energy comparison.

(0.031) higher score, but for datasets with more features, the proposed GENERIC can better benefit from using windows (windows become less effective in a smaller number of features).

Figure 4.9 compares the per-input energy consumption of GENERIC with K-means clustering running on CPU and Raspberry Pi. GENERIC consumes only 0.068 μJ per input, which is 17,523$\times$ and 61,400$\times$ more efficient than K-means on Raspberry Pi and CPU. The average per-input execution time of Raspberry Pi and CPU is, respectively, 394 μSec and 248 μSec, while GENERIC achieves 9.6 μSec (41$\times$ and 26$\times$ faster than R-Pi and CPU, respectively).

Table 4.2: Mutual information score of K-means and HDC.

	Hepta	Tetra	TwoDiamonds	WingNut	Iris
K-means	1.0	0.637	1.0	0.774	0.758
HDC	0.904	0.589	0.981	0.781	0.760

4.6 Conclusion

We proposed GENERIC, a highly-efficient HDC accelerator that supports classification (inference and training) and clustering using a novel encoding technique that achieves 3.5% (6.5%) better accuracy compared to other HDC (ML) algorithms. GENERIC benefits from power-gating, voltage over-scaling, and dimension reduction for utmost energy saving. Our results showed that GENERIC improves the classification energy by 15.1× over a previous trainable HDC accelerator, and 4.1× over an inference-only accelerator. Compared to random forest (most-efficient conventional ML), GENERIC consumes 528× and 1593× less training and inference energy, respectively, with 8.2% higher accuracy. Compared to DNNs running on eGPU, GENERIC reduces the training energy by 1257×, inference energy by 6230×, while achieves 1.0% better accuracy. GENERIC HDC-based clustering consumes 17,523× lower energy with 41× higher performance than Raspberry Pi running K-means with similar accuracy.

So far, we attempted to address the energy efficiency of edge intelligence by proposing FPGA and ASIC that rely on HDC algorithm. In the next chapter, we show how noise robustness of HDC can be leveraged to realize privacy-preserved and efficient inference and training at the edge.

Chapter 4, in part, is a reprint of the material as it appears in "GENERIC: Highly Efficient Learning Engine on Edge using Hyperdimensional Computing" by Behnam Khaleghi, Jaeyoung Kang, Hanyang Xu, Justin Morris, and Tajana Rosing, which appears in Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC), 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Private Learning and Inference with Hyperdimensional Computing

In Chapter 3 and Chapter 4 we showed how the simple algorithm and operations of HDC can bring orders of magnitude energy savings for learning tasks. In this chapter, we leverage HDC to cope with the privacy issues of machine learning algorithms at the edge. We show that HDC can offer privacy-preserved training and inference with comparable or better accuracy than DNNs for many workloads, including larger size image datasets, with significantly faster training time.

5.1 Introduction

The number and diversity of applications that can take advantage of machine learning (ML) algorithms to gain insightful information from data continue to grow [21, 23]. However, sensory and edge devices have tight resource and energy constraints or operate in error-prone environments [24, 26, 27]. One approach to tackle this issue is to offload the computation to

gateway or cloud [14, 123, 124]. Nevertheless, transmitting the data, particularly in sensory devices that cannot accommodate complex encrypted schemes, can expose it to an untrustworthy communication link or server.

Previous studies have attempted to obfuscate the transmitted information to address the privacy breach of cloud-hosted inference of DNNs. These techniques mainly run the initial layers of the network on the edge device and transmit the noise-injected (obfuscated) features to carry on the rest of the operations on the cloud [125]. However, running part of the compute- and memory-heavy networks on the edge device is not feasible for many edge devices [26]. In addition, DNN layers expand the size of data compared to the raw input [126], which aggravates the communication cost.

Another challenge with the privacy of DNNs is membership inference, i.e., whether a particular sample was used during the model training. To deal with the training privacy, previous works have leveraged *differential privacy* (DP) [43, 125]. By randomizing the output of a mechanism by adding a particular noise, DP guarantees that the output does not reveal the information of individual records. As an example, consider the *sum* query, denoted by $\mathcal{M}$, on a database. Although $\mathcal{M}$ does not directly reveal the value of a particular row r, it can be inferred by $\mathcal{M}(\mathcal{D})-\mathcal{M}(\mathcal{D}-\{r\})$, i.e., calling $\mathcal{M}$ twice on certain rows with and without r. DP increases the cost of revealing r by adding a random noise on top of $\mathcal{M}(\mathcal{D})$, so more queries are needed to cancel out the noise. In the context of ML training, a network trained over two training datasets that differ in one sample (called *adjacent* datasets) should not reveal the input when the models are inspected. However, DNN training is a sensitive mechanism and adding noise on the trained models impairs the training convergence and leads to poor accuracy [125].

In this chapter, we use Hyperdimensional Computing (HDC) to cope with the privacy issues of edge computing. HDC is a new learning paradigm that addresses the efficiency and reliability limitations of conventional ML algorithms. HDC encodes raw input data to points in hyperspace, represented by vectors of thousands of bits, which makes it robust to error as

information is distributed over the components of a vector [127, 128]. Training using HDC is as simple as adding up the vectors of the same label to create the class vectors. For inference, the query is encoded in the same fashion and compared with the class vectors using simple metric such as Hamming distance for binary vectors, or dot-product. All these operations are low-precision and parallelizable, which makes custom implementations of HDC more than two orders of magnitude more energy efficient than the other ML counterparts [40, 129].

HDC, however, does not inherently preserve privacy. Encoding is reversible and thus can return the original data as a part of the decoding process [3, 130]. Therefore, in applications such as federated learning or cloud-hosted inference when the model does not fit on edge device memory [42, 41], transmitting the encoded data jeopardizes the privacy as an untrustworthy link or server can reconstruct the original data. Moreover, as the HDC classes are built by simply superposition of vectors, small changes in the training data can reveal the statistics of the particular encoded vectors, which can result in the membership breach.

That being said, we show that the superior error tolerance of HDC [127, 128] can be prudently leveraged to enhance the privacy by noise/error injection. Specifically, we make the following novel contributions.

(1) We demonstrate the reversibility of common encoding methods, which indicates that HDC is vulnerable to privacy breach. We also show that the encoding parameters that are required for HDC decoding can be extracted by using adversarial inputs, hence obscurity alone does not guarantee privacy (Section 5.3).

(2) We propose novel hardware and efficient communication techniques to improve the HDC inference privacy by reducing the information of the transferred data through approximate locally-sparse encoding (Section 5.4).

(3) We propose privacy-preserving one-pass and iterative training of HDC to combat membership inference breaches (Section 5.5).

(4) We propose Privé-HDnn, a combination of CNN feature extractor and HDC classifier for

privacy preserving image classification (Section 5.5).

5.2 Related Work

The closest study related to our work is Prive-HD [3], which trains an HDC model and adds a Gaussian noise to the final model. Prive-HD assumes that the trained models of adjacent datasets are different only in one vector (i.e., the encoded vector of the extra record) and sets the amount of noise injection accordingly. While this works for one-pass training, in iterative training the updates in the subsequent epochs depend on the initial model. Hence, one extra sample changes the initial model which in turn affects the subsequent updates and can make the trained models significantly different. Likewise, the randomness of the training batches also impacts the final model. Considering only a *single* vector difference is a substantial underestimate of the required noise.

PRID [130] proposes model inversion attack and defense techniques on HDC models. It first decodes the model classes into the original space, which represents rough estimates of the utilized training data. Then, given a query, PRID attempts to reconstruct the train data by replacing a subset of input query features (e.g., pixels of an image) with the features of the decoded class so that the similarity score increases. PRID uses dimension reduction and model quantization to reduce information leakage.

SecureHD [42] aims to protect user data from an untrustworthy cloud system in a collaborative learning system. The idea is that each client has distinct base vectors to encode the data, which are generated by shuffling a seed base. The cloud has only the shuffling keys (not the bases) to reshuffle the clients' encoded vectors; hence, after reshuffling, all the vectors are encoded by the same bases, and HDC learning becomes possible. Nonetheless, such obscurity does not guarantee privacy. As we show in this chapter, the private encoding parameters (base vectors in [42]) can be revealed from the encoded vectors through adversary inputs. Moreover, an

adversary client can share its private key with the cloud, by which the cloud can reconstruct the seed bases (hence, all the private bases) using the shuffling keys.

Another body of research has targeted various attacks on HDC. The study in [131] uses genetic algorithm to generate adversarial images with minimal perturbation that leads to misclassification by the model. HDTest [132] applies random row/column mutation, shift, and noise to generate adversarial images. PoisonHD [133] aims to degrade the performance of the HDC model by injecting false data with flipped labels.

5.3 Reversibility of HDC

In this section, we introduce the common encoding methods and show how the original data can be reconstructed from the encoding vectors to jeopardize privacy. HDC decoding process needs the constant vectors that are used for encoding. We also show that these parameters (encoding vectors) can be discovered by using adversarial inputs, so privacy cannot be attained merely through obscurity.

5.3.1 Encoding and Decoding

Figs. 4.1(a)-(c), illustrate the common HDC encoding techniques, each of which preserves a particular distance in the hyperspace [31] and is suitable for particular type of data [129]. We represent input samples as a d element feature vector $V = \langle v_0, v_1, \cdots, v_{d-1} \rangle$, and we use capital D for encoded vectors dimensionality. We use vector symbol$\vec{}$ to represent vectors in the hyperspace.

(1) Random projection (RP) encoding, shown in Figure 4.1(a), multiplies each feature with a D-dimensional base vector associated with that index. $\vec{\mathcal{B}}_k$s are constant orthogonal vectors called *base* or *id* vectors that are used to preserve (bind) the spatiotemporal relations of the features. RP encoding can be transformed to a matrix-vector multiplication by laying the d base

vectors as the columns of a $D \times d$ matrix $\mathcal{B}^T$.

$$\vec{\mathcal{H}} = \sum_{k=0}^{d-1} v_k \times \vec{\mathcal{B}}_k = \mathcal{B}^T \times V \tag{5.1}$$

Thus, we can decode $\vec{V}$ in the following way:

$$(\mathcal{B}^T)^\dagger \times \vec{\mathcal{H}} = (\mathcal{B}^T)^\dagger \times \mathcal{B}^T \times V \Rightarrow V \simeq (\mathcal{B}^T)^\dagger \times \vec{\mathcal{H}}$$

Since $\mathcal{B}$ is non-square, we use Moore-Penrose pseudo-inverse [134] to estimate $\mathcal{B}^\dagger$.

(2) Base-level (aka *id-level*) encoding, instead of directly using the raw values of features, uses a set of high-dimensional level vectors $\vec{\mathcal{L}}$ to represent values. Both the base and level vectors have bipolar components, i.e., $\{+1,-1\}^D$. For q level vectors, $\vec{\mathcal{L}}_0$ is generated randomly, then every $\vec{\mathcal{L}}_k$ is generated by flipping $\frac{D}{2q}$ bits of $\vec{\mathcal{L}}_{k-1}$. Therefore, closer features have more similar levels and vice versa; particularly, $\vec{\mathcal{L}}_0 \cdot \vec{\mathcal{L}}_{q-1} \simeq 0$ (e.g., white and black pixels have the most different level vectors). The number of level vectors is limited, so input values are quantized to q bins to obtain the right level of each value. Equation (5.2) formulates the base-level encoding:

$$\vec{\mathcal{H}} = \sum_{k=0}^{d-1} \vec{\mathcal{L}}(v_k) \times \vec{\mathcal{B}}_k \tag{5.2}$$

We can decode the base-level encoding index by index. To obtain a feature v_i, we first multiply $\vec{\mathcal{H}}$ by $\vec{\mathcal{B}}_i$ so we have:

$$\vec{\mathcal{B}}_i \times \vec{\mathcal{H}} = \vec{\mathcal{B}}_i \times \sum_{k=0}^{d-1} \vec{\mathcal{L}}(v_i) \times \vec{\mathcal{B}}_k = \vec{\mathcal{L}}(v_i) + \sum_{\substack{k=0 \\ k \neq i}}^{d-1} \vec{\mathcal{L}}(v_k) \times \vec{\mathcal{B}}_k$$

Then, we try $\vec{\mathcal{L}}_x$ on the resultant vector to find out the one with highest dot-product:

$$\vec{\mathcal{L}}_x \cdot (\vec{\mathcal{B}}_i \times \vec{\mathcal{H}}) = \vec{\mathcal{L}}_x \cdot \vec{\mathcal{L}}(v_i) + \overbrace{\vec{\mathcal{L}}_x \cdot \sum_{\substack{k=0 \\ k \neq i}}^{d-1} \vec{\mathcal{L}}(v_k) \times \vec{\mathcal{B}}_k}^{noise(\approx 0)} \tag{5.3}$$

Since base vectors are random, expected value of the right-hand side of the above equation is ≈ 0, and $\vec{\mathcal{L}}_x \cdot \vec{\mathcal{L}}(v_i)$ becomes maximum when $x = i$. Once we obtained $\vec{\mathcal{L}}(v_i)$, we can infer v_i from the associated level.

(3) Permutation uses a different way of binding the spatiotemporal information. Instead of using base vectors, it applies $P^{(k)}$ on the level vector of each feature v_k, where $P^{(k)}\left(\vec{\mathcal{L}}\right)$ denotes circular shift of $\vec{\mathcal{L}}$ by k indexes. It can be formulated as follows:

$$\vec{\mathcal{H}} = \sum_{k=0}^{d-1} P^{(k)}\left(\vec{\mathcal{L}}(v_k)\right) \tag{5.4}$$

Decoding the permuting encoding can be done in a similar way to base-level decoding, except for multiplying by $\vec{\mathcal{B}}_i$ in the first step, we permute $\vec{\mathcal{H}}$ by $-i$ indexes, then try to find the $\vec{\mathcal{L}}_x$ that maximizes the dot-product.

5.3.2 Parameter Extraction

In the previous subsection, we showed how the vectors of common encoding methods can be decoded to the original data. Decoding the vectors needs knowledge of the HDC *parameters*, i.e., the base and/or level vectors. Here, we show how these parameters can be extracted by using adversary inputs. Previous studies Prive-HD [3] and PRID [130] decode the random projection encoding only and assume prior knowledge of the encoding parameters. *Hence, our work is the first one that introduces various decoding technique and shows hot to extrat their parameters.* Parameter extraction can break the security mechanisms such as SecureHD [42] that build on the

assumption that these parameters remain unknown. We consider a gray-box model where the input format and encoding algorithm is known, but not its parameters, and we can observe the encoded outputs. This is a plausible scenario in federated learning or cloud inference [42] where the clients send the encoded outputs to the cloud.

(1) RP parameter extraction. Random projection is a simpler encoding as it directly projects each scalar input feature by multiplying it with the associated base vector. By feeding an adversary input $V = e^{(i)}$, i.e., a null input vector except $v_i = 1$ in Equation (5.1), we have:

$$\vec{\mathcal{H}} = \sum_{k=0}^{d-1} v_k \times \vec{\mathcal{B}}_k = 1 \times \vec{\mathcal{B}}_i + \sum_{\substack{k=0 \\ k \neq i}} 0 \times \vec{\mathcal{B}}_k = \vec{\mathcal{B}}_i$$

That is, to infer the i^{th} base $\vec{\mathcal{B}}_i$, we only need to generate an adversary input with i^{th} feature 1 and others 0. The observed encoded output $\vec{\mathcal{H}}$ is the same as $\vec{\mathcal{B}}_i$. Our approach requires d adversary inputs to extract all d base vectors.

(2) Permutation parameter extraction. To extract the level vectors, we first need to find the number and size (length) of the quantization bins. To this end, we create an all-equal-features adversary input $V = \{\alpha\}^d$ starting with the lowest value for α based on the application. By gradually increasing α, the encoded vector $\vec{\mathcal{H}} = \sum_{k=0}^{d-1} P^{(k)}\left(\vec{\mathcal{L}}_0\right)$ remains the same until α falls in the next bin ($\vec{\mathcal{L}}_1$). By continually increasing the α, we can obtain the number and length of the bins.

Then, to extract the x^{th} level vector, we generate an adversary input in which all features fall in the x^{th} bin. Having $\vec{\mathcal{L}}_x = \{x_0, \cdots, x_{D-1}\}$, Figure 5.1 shows an example where $d = 4$ features and $D = 6$ dimensions, hence, $\vec{\mathcal{H}}^4 = \vec{\mathcal{L}}_x^4 + \vec{\mathcal{L}}_x^5 + \vec{\mathcal{L}}_x^0 + \vec{\mathcal{L}}_x^1 = x_4 + x_5 + x_0 + x_1$. In general, it creates a system of D linear equations with D unknowns, where each equation has d coefficients of $'1'$, and the rest $D-d$ of $'0'$, and has a unique solution given that $\vec{\mathcal{L}}_x \in \{+1, -1\}^D$. We need only one adversary input to infer each level vector. While we assumed a typical binding where the level vector at index k is permuted by k, our approach works for arbitrary permutations; we still get a

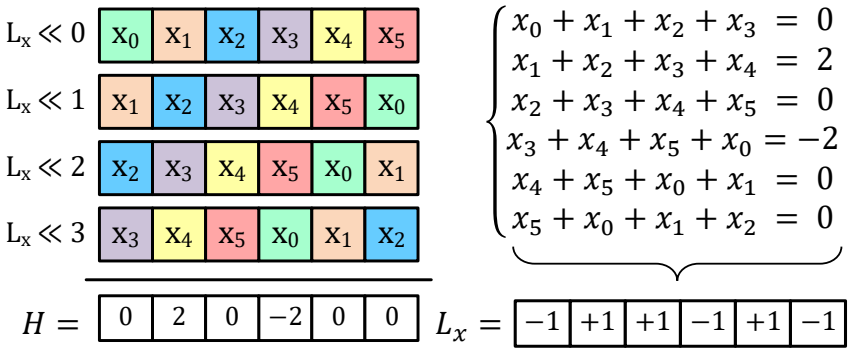


Figure 5.1: An example of inferring an unknown level vector $\vec{L}_x$ by setting all four features v_k to $\vec{L}(v_k) = \vec{L}_x$.

system of D linear equations.

(3) Base-level parameter extraction. Here we deal with two unknowns, level and base vectors. We aim to first infer the level vectors, followed by the base vectors. We denote the inferred level and base vectors by $\vec{L}'$ and $\vec{B}'$. Base-level extraction does not have a unique solution, so we set $\vec{L}'_0 = \{1\}^D$. Similar to permutation case, we use an adversary $V = \{\alpha\}^d$ and increase α until $\vec{\mathcal{H}}$ changes. Once α falls in the next bin ($\vec{L}_1$), a j^{th} component of encoded $\vec{\mathcal{H}}$ changes only if the same component is different in $\vec{L}_0^j$ versus $\vec{L}_1^j$. As we set $\vec{L}'_0 = \{1\}^D$, we observe the changed components of $\vec{\mathcal{H}}$ and set those of $\vec{L}'_1$ to -1. We keep increasing the α to create $\vec{L}'_2$ upon the next change of $\vec{\mathcal{H}}$ and so on.

Figure 5.2 shows an example with $D = 6$ dimensions, $d = 3$ features/bases and $m = 4$ levels/bins. ❶ and ❷ are the original level and base vectors. In ❸, we have observed all the $m = 4$ encoded vectors by using inputs in the form of $V = \{\alpha, \alpha, \alpha\}$. The yellow elements show changes of $\vec{\mathcal{H}}$ as α increases. In ❹, we set $\vec{L}'_0 = \{1\}^6$ and generate the rest of the levels by comparing the $\vec{\mathcal{H}}_1$, $\vec{\mathcal{H}}_2$ and $\vec{\mathcal{H}}_3$ with $\vec{\mathcal{H}}_0$. The inferred $\vec{L}'$ is different from $\vec{L}$ in dimensions shown in red.

After inferring the $\vec{L}'$ vectors, we create a system of linear equation for each index, with $\vec{L}'$ and the observed $\vec{\mathcal{H}}$ as constant, and $\vec{B}'$ as variable. Figure 5.2❻ shows the equations corresponding to $\vec{B}'^1$ (index 1 of bases). We decided to use d adversary inputs in the form

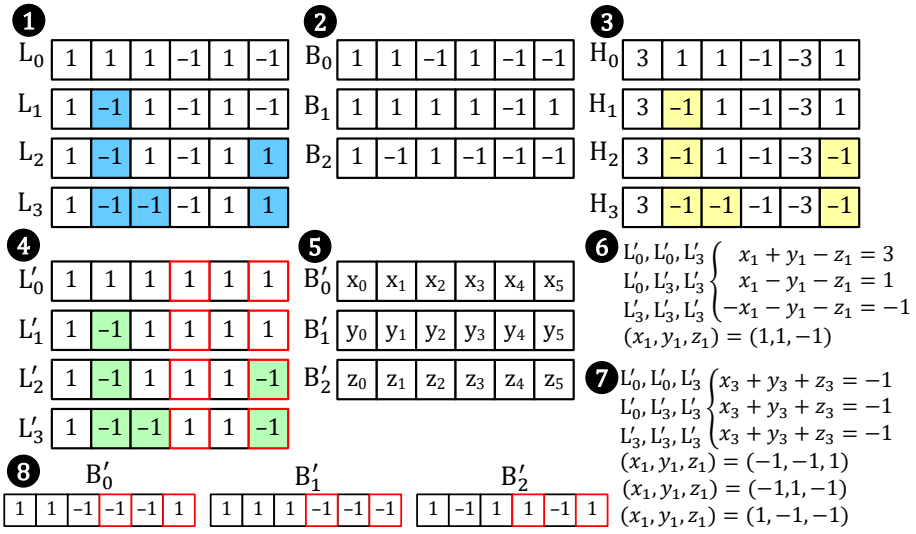


Figure 5.2: An example of extracting the level and base vectors.

of $V_1 = \{v_{min}, \cdots, v_{min}, v_{max}\}$, $V_2 = \{v_{min}, \cdots, v_{max}, v_{max}\}$, $\cdots$, and $V_d = \{v_{max}, \cdots, v_{max}, v_{max}\}$ to assure the rows are linearly independent. We need only d adversary inputs to infer the bases, the same inputs that are used to simultaneously infer all indexes. Figure 5.2 ❽ shows the inferred base vectors $\vec{\mathcal{B}}'$, which are different from $\vec{\mathcal{B}}$, but $\{\vec{\mathcal{L}}', \vec{\mathcal{B}}'\}$ returns the same encoding as the original $\{\vec{\mathcal{L}}, \vec{\mathcal{B}}\}$.

5.4 Privacy-Preserved HDC Inference

We improve the privacy of HDC inference by obfuscating the information of offloaded encoded vectors, so that the reconstructed data is indiscernible. We rely on noise tolerance of HDC [127, 128] and make the vectors highly sparse by keeping the most effectual elements only. The sparsification is done in a regular manner; each segment of the vectors is sparsified independently, which helps in efficient hardware implementation and simple index-based compression for communication purpose as we detail in the following.

(1) Local sparsification: HDC has performed well in cloud-hosted inference [42] and federated learning where it outperforms CNNs by having significantly smaller communication

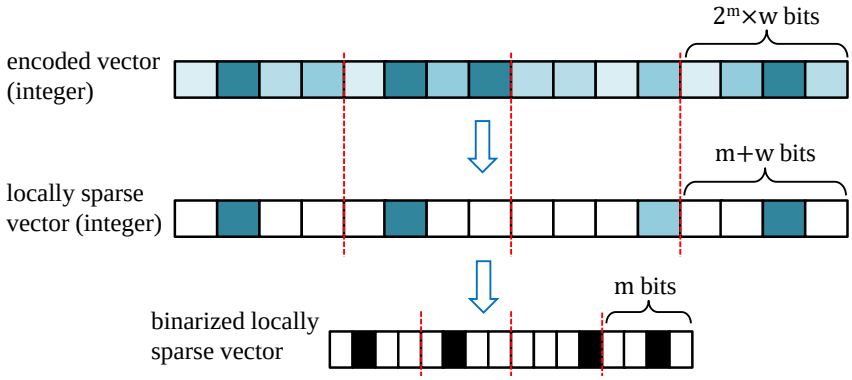


Figure 5.3: Local sparsification. Darker colors indicate larger values.

cost, consuming less local compute energy [41], and being more robust to noise [128]. All these use cases rely on transmitting the encoded vectors to the cloud. However, the reversibility of HDC encoding that we demonstrated in the previous section poses a serious privacy challenge when the communication link or the cloud is untrustworthy.

To reduce the exposed information as a result of decoding by an adversary link or cloud, we use sparse representation of HDC. Sparsification can be deemed as a lossy compression that reduces the recoverable information. A plausible approach for sparsification is to keep the top-k components of the encoded vector and zero-out the rest. This is conceivable as these components have higher contribution to the similarity score (dot-product), and concurs with the sensory systems of many organisms such as the olfactory system of fruit fly that perform a winner-take-all thresholding on the expanded representation of the input odor [31, 135].

Our goal is to realize sparsification in a hardware and communication efficient fashion. Selecting top-k of the encoded components is trivial in a processor via sort operation. However, sort is irregular and not done efficiently in low-power devices [136]. Instead of selecting the top-k percentage of components over the entire vector, we find the local maximums and sparsify *locally*. Figure 5.3 demonstrates the concept. For a target non-sparse rate of $\frac{1}{2^m}$, we divide the encoded vector into segments of 2^m elements, and replace all but the maximum element of each segment with zero.

(2) Efficient FPGA implementation: Since we are interested only in the *index* of the maximum components rather than their exact values, we can use more-efficient approximate encodings to obtain the components. For base-level and permutation encodings that vastly use binary popcount for bundling, in a previous work [2] we proposed approximate binary popcounts that simplify these encodings by replacing part of the sum operations with majority functions (Figure 5.4(a)). Using majority function reduces up to of 80% FPGA resources, and although the encoding components are approximate, they are suitable in the context of sparsification where only the relative value of components matters.

That being said, here for focus on a novel approximate implementation for locally-sparse *random projection* encoding. Figure 5.4(b) shows an FPGA-friendly implementation of the proposed sparse RP encoding. RP multiplies the projection matrix with the input vector. The idea is to bypass some matrix elements when generating the encoding. We split the original matrix into partitions of 2^m rows and n columns each, with $\frac{1}{2^m}$ non-sparse rate. Multiplication of a whole row of the matrix with the input vector produces one output component. Among these 2^m rows, the output of only one of them will be 1 and the remaining will be zeroed out. At each step, one bit of a partition's column is multiplied with the corresponding feature. The partial encoding result compares the adjacent values, which belong to adjacent rows, and allows the maximum one to proceed to the next step. In Figure 5.4(b), the red bits are for the first step, active in all rows, while the green bits are activated for the rows that passed to the second step. Accordingly, m steps are needed to finalize the encoding.

We choose $m \leq n$, which means that even for a row that proceeds to till the last step, $n-m$ bits of it still remains unused. That is, the projection matrix is also sparse. The sparsity pattern varies among the partitions so none of the matrix columns is entirely zero. This approach does not require local sorting as, at each step, adjacent outputs are compared and the maximum index is passed to to the next steps. This helps to realize a deterministic and hardware (e.g. FPGA) friendly data flow.

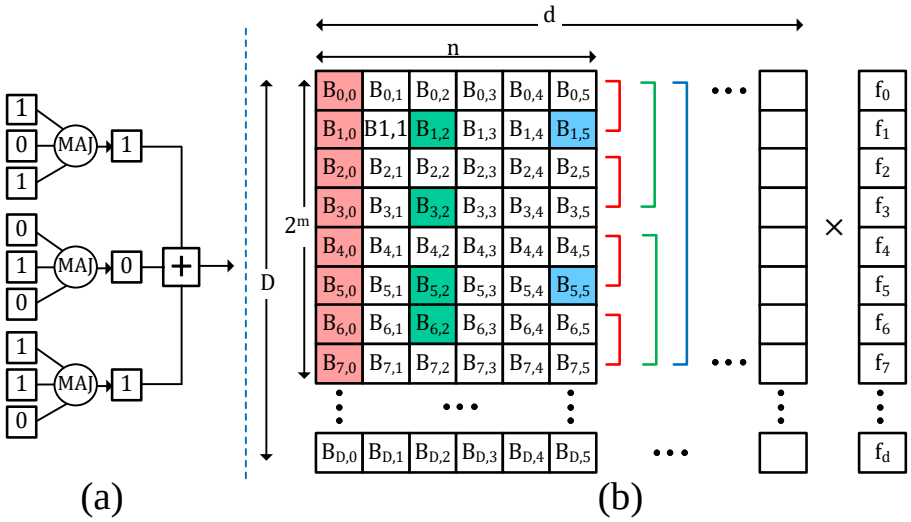


Figure 5.4: (a) Approximate element-wise addition, suitable for sparse base-level and permutation encodings [2], (b) approximate local sparse encoding random projection.

(3) Communication compression: One advantage of our approach is in reducing the communication overhead. Instead of sending $2^m \times w$ bits per segment (2^m components, w bit each), we just need to send $m+w$ bits, i.e., m bits for the non-zero index and w bits for its value. Since HDC encodings work well with quantization [103], we can further quantize the encoding vectors down to binary. Hence, only m bits per segment is needed. In normal top-k sparsification, besides the complication of finding top-k, we would need $\log(D)$ bits per non-zero component. We quantify the improvement of our approach in Section 5.6.

5.5 Differentially Private HDC Training

In contrast to DNNs that are built from more abstract backpropagation parameters, HDC models are built by superposition of encoded data, which, as we showed, can be decoded to the original data. Thus, the privacy of HDC training is more challenging as the data inputs are mapped onto a reversible vector. In this section, we show how differential privacy (DP) can be applied to protect the data privacy of HDC for both one-pass and iterative (multi-pass) training

using state of the art HDC encoding techniques discussed in previous section.

While most classification applications have excellent accuracy when encoding raw data into HDC vectors, our recent work showed that for larger size image data, such as CIFAR 100, we need to use a subset of a pretrained neural network as a feature extractor prior to HDC-based encoding and training [137]. To solve privacy problems for such HDC-based image classifiers, we propose a hybrid CNN-HDC approach, dubbed Privé-HDnn, that enables differentially private yet accurate image classification.

As we explained in Section 5.2, previous work [3] targeted private iterative HDC training by post-ex noise injection. However, as we show in Section 5.5.3, [3] substantially underestimates the amount of additive noise. In fact, post-training noise injection is not a viable approach for iterative training.

5.5.1 Differential Privacy (DP)

A randomized algorithm $\mathcal{M}$ with domain $\mathcal{D}_{\mathcal{M}}$ is (ε, δ)-differentially private if for any two adjacent datasets $\mathcal{D}, \mathcal{D}_{-1} \in \mathcal{D}_{\mathcal{M}}$ that differ in one record (an input sample), for all output subset $\mathcal{S}$ of $\mathcal{M}$ the following holds [43]:

$$Pr[\mathcal{M}(\mathcal{D}) \in \mathcal{S}] \leq e^{\varepsilon} Pr[\mathcal{M}(\mathcal{D}_{-1}) \in \mathcal{S}] + \delta$$

which means that observing $\mathcal{D}$ after $\mathcal{D}_{-1}$ increases the probability of an event by no more than e^{ε}. The additive term δ allows breaking the DP by a probability of δ, which is usually selected to be smaller than $\frac{1}{|\mathcal{D}|}$.

A common approach to satisfy DP is applying a Gaussian noise proportional to the *sensitivity* of the algorithm, defined as the maximum amount of change of the output if the input differs in one element. For Gaussian noise, we use ℓ_2 norm for sensitivity, i.e., $\Delta_{\mathcal{M}} =$

$\|\mathcal{M}(\mathcal{D}) - \mathcal{M}(\mathcal{D}_{-1})\|_2$. Thus,

$$\mathcal{M}_{DP}(\mathcal{D}) = \mathcal{M}(\mathcal{D}) + \mathcal{N}(0, \Delta_{\mathcal{M}} \cdot \sigma) \tag{5.5}$$

in which $\Delta_{\mathcal{M}} \cdot \sigma$ is the standard deviation of the noise. For a fixed δ and a privacy target ε, the parameter σ can be obtained to satisfy $\delta \geq \frac{4}{5} e^{-\frac{(\sigma\varepsilon)^2}{2}}$ [43]. A smaller ε or δ (i.e., less likelihood of leakage) demands larger noise parameter σ.

5.5.2 Private One-pass Training

One-pass HDC training simply accumulates the encoded vectors of the same label as follows below.

$$\vec{C}_j = \sum_{i \text{ s.t.} y_i = j} \vec{\mathcal{H}}_i \tag{5.6}$$

When an input V is discarded from the dataset, only one of the class vectors is affected, where $\vec{C}_{-1} = \vec{C} - \vec{\mathcal{H}}(V)$. Thus, the sensitivity of one-pass model is $\Delta_{\mathcal{M}} = \| \vec{\mathcal{H}}(V) \|_2$. This is a valuable property as it shows the sensitivity of one-pass training does not depend on the dataset size. We can train a one-pass HDC model on arbitrary datasets and add a noise based on $\Delta_{\mathcal{M}_{max}}$ which is independent of the encoding and the dataset. For inference, we will have ($\vec{C}'$ are classes with additive noise):

$$y = \underset{i \in C}{\text{argmax}}\, \vec{C}'_i \cdot \vec{\mathcal{H}} = \underset{i \in C}{\text{argmax}}\, \vec{C}_i \cdot \vec{\mathcal{H}} + \vec{\mathcal{N}} \cdot \vec{\mathcal{H}} \tag{5.7}$$

A large noise can impact the $\vec{\mathcal{N}} \cdot \vec{\mathcal{H}}$ and change the scores rank. Each encoded component $\vec{\mathcal{H}}^j$ belongs to $[-d, d]$ (with $v_{max} = 1$). This implies a sensitivity of $\Delta_{\mathcal{M}_{max}} = d\sqrt{D}$ that demands a significant amount of noise $\mathcal{N}\big(\mu = 0, \sigma = \frac{\Delta_{\mathcal{M}_{max}}}{\varepsilon} \sqrt{2\log(\frac{1.25}{\delta})}\big)$ that can be as large as the class values themselves. It is because DP considers the worst-case of $\vec{\mathcal{H}}$ where components are $\{\pm d\}$.

To solve this issue, we need to ensure that $\vec{\mathcal{H}}$ does not exceed an expected bound, so we can use a smaller noise in accordance. Thus, we normalize and clip the encoded vectors by

$\vec{\mathcal{H}}' = \vec{\mathcal{H}} / \max(1, \frac{\|\vec{\mathcal{H}}\|_2}{\kappa})$ which ensures $\|\vec{\mathcal{H}}'\|_2 \leq \kappa$. By choosing, e.g., $\kappa = 1$, we achieve $\Delta_{\mathcal{M}_{max}} = 1$ which shrinks the variance of noise. By normalizing the encoded vectors, the values of class vectors also scale accordingly, hence, normalization alone cannot mitigate the impact of noise; clipping bounds the values and helps to avoid worst-case scenarios and paying extra sensitivity ($\Delta_{\mathcal{M}}$) cost.

5.5.3 Private Iterative Training

Iterative (multi-pass) HDC training improves the HDC accuracy by evaluating the model on the training data. In case of wrong prediction, the encoded vector is subtracted from the mispredicted class to make them less similar, and is added to the expected class once again as follows below.

$$\begin{aligned} \vec{C}_f &= \vec{C}_f - \vec{\mathcal{H}} \\ \vec{C}_t &= \vec{C}_t + \vec{\mathcal{H}} \end{aligned} \tag{5.8}$$

Privacy of iterative training cannot simply be obtained by finding the sensitivity and adding the post-training noise. In iterative training, a discarded or extra input affects the model updates in the subsequent epochs. Normalizing the encoded vectors guarantees $\Delta_{\mathcal{M}_{max}} = \|\vec{\mathcal{H}}\|_2 \leq \kappa$ for one-pass learning, but it does not hold for iterative training. The work in [3] was built on this assumption. For the datasets of Section 5.6, after normalizing the vectors to $\Delta_{\mathcal{M}_{max}} = \|\vec{\mathcal{H}}\|_2 \leq 1$, we observed that excluding only one input makes $\Delta_{\mathcal{M}}$ of $[\sim 20-43]$ which makes the required noise σ intractable.

To realize differentially private multi-pass training, we use the composition property of DP. A mechanism with a series of ε_i-private steps is $\sum\varepsilon_i$-differentially private [138]. Thus, we can preserve privacy at the batch level, where the total privacy cost of training is the total cost of the batches. It helps by requiring smaller noise, and although the noise will be injected frequently

Algorithm 4: Differentially private iterative HDC training

Input: Dataset $\mathcal{D} = \{\vec{V}_1, \cdots \vec{V}_N\}$, batch size L, noise variance σ^2, encoding norm-2 limit κ

Output: Class vectors $\vec{C} = \{\vec{C}_1, \cdots \vec{C}_C\}$

1 $\vec{\mathcal{H}}_{\mathcal{D}} \leftarrow \text{enc}(\mathcal{D})$ // encode the train data
2 $\vec{C} \leftarrow 0$
3 **for** t *in* $[1:T]$ **do**
4 $\vec{G}[1:C] = \{0\}^D$ // initialize class update vectors per iteration
5 $L_t \leftarrow \text{random\_select}(\vec{\mathcal{H}}_{\mathcal{D}}, L)$ // $L=1$ in normal training
6 **for** $\vec{\mathcal{H}}$ *in* L_t **do**
7 $\ell = \text{argmax}_i \vec{C}_i \cdot \vec{\mathcal{H}}$ // inference on the train sample
8 **if** $\ell \neq \ell_{\mathcal{H}}$ **then**
9 $\vec{\mathcal{H}}' \leftarrow \vec{\mathcal{H}} / \max(1, \frac{\sqrt{2}\|\vec{\mathcal{H}}\|_2}{\kappa})$ // bound the sensitivity
10 $\vec{G}[\ell] \leftarrow \vec{G}[\ell] - \vec{\mathcal{H}}'$ // subtract from mispredicted class ℓ
11 $\vec{G}[\ell_{\mathcal{H}}] \leftarrow \vec{G}[\ell_{\mathcal{H}}] + \vec{\mathcal{H}}'$ // add to the golden class $\ell_{\mathcal{H}}$
12 $\vec{G} \leftarrow \vec{G}/L + \vec{\mathcal{N}}(0, \kappa \cdot \sigma)$ // average the updates and add noise
13 $\vec{C} \leftarrow \vec{C} + \vec{G}$ // update the classes
14 **return** $\vec{C}$

per batch, the model has the opportunity to be calibrated against the introduced noise. If the training is divided into batches of $L = qN$ randomly selected samples (for N being the dataset size), following the privacy amplification theorem, each iteration is $(q\varepsilon, q\delta)$-private with respect to the whole dataset [139, 43]. For T training iteration ($E = q \times T$ epochs), [43] provides a tighter bound of $(q\varepsilon\sqrt{T}, \delta)$ on privacy of the composite model with the following additive noise:

$$\sigma \geq c \frac{q}{\varepsilon} \sqrt{T \ln(1/\delta)} \tag{5.9}$$

Algorithm 4 summarizes the composite private learning for HDC. The lines with comments in red are exclusive to the private training. At each of T training iterations, a random batch of encoded data is selected (line 5). Similar to DNN training, a forward inference pass is run over the batch (line 6–7). For the data with mispredicted labels (line 8), the normalized vectors

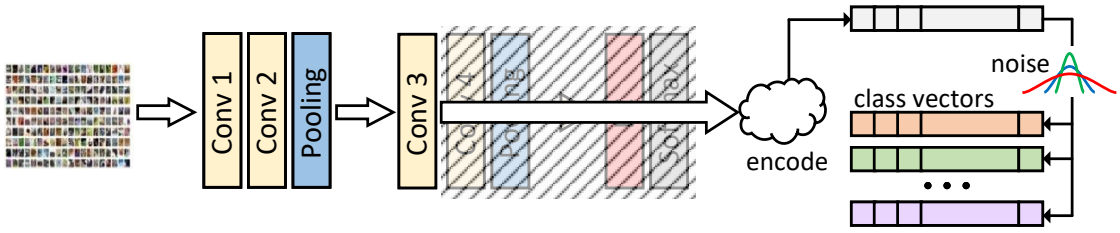


Figure 5.5: Privé-HDnn for private training on images. A shallow CNN extracts the features (only once) as raw inputs for HDC, which performs private training according to Algorithm 4.

are saved (similar to gradients of different samples in neural networks), and averaged followed by noise injection (line 14). The algorithm is fairly similar to normal HDC training, except the updates are done per a batch for L samples, as opposed to per sample, and the encoded vectors are normalized to bound the worst-case sensitivity as discussed in subsection 5.5.2.

5.5.4 Privé-HDnn: Private Image Classification

HDC achieves poor accuracy for complex image datasets when using state of the art HDC encoding directly on the raw image data. A way to resolve this issue is to train a CNN-based feature extractor and use HDC for classification over the extracted features [137, 41]. The hybrid architecture is called HDnn. The conventional HDnn flow consists of (i) extracting image features using a subset of a pretrained CNN (ii) training an HDC model over the extracted features, and (iii) attaching the trained HDC to the CNN and calibrating the CNN to compensate potential accuracy loss. HDnn offers other advantages such as few-shot learning [140] due to capability of HDC in learning from fewer data, and more straightforward in-field learning as only the HDC head needs to be updated.

We leverage the HDnn structure and propose Privé-HDnn, shown in Figure 5.5, for private learning on images. A major issue with CNNs is their intolerance to additive noise due to the highly sensitive gradient descent backpropagation. Previous studies on privacy-preserving training of CNNs have shown low accuracies, e.g. 72% on `CIFAR-10` [141]. Some of the previous

works use simple networks and even keeps some of the layers fixed [43].

We circumvent the noise intolerance of DNNs by using transfer learning over a *public dataset* from a different distribution. We keep the pretrined CNN feature extractor unchanged. That is, we *do not* calibrate it on the target dataset after attaching the HDC. Hence, it acts as a constant function and we do not need to add noise on the CNN part as does not learn nor expose any information of the target dataset. Thus, CNN retains its exact functionality.

The HDC classifier head, however, is trained on the extracted features of the target dataset with differential privacy using the proposed iterative method of Algorithm 4. The extracted features act as raw data for HDC algorithm. Therefore, data is encoded and class vectors are realized by bounding the sensitivity of encoded vectors, followed by adding noise before bundling. We use a ResNet-18 network pretrained on ILSVRC 2012 dataset as the constant function. For privacy purposes, we do not calibrate the public model after attaching the HDC in place of the last layers. For efficiency purpose – which is not the main focus of this work – any shallower model could replace ResNet-18.

5.6 Results

(1) Setup: In this section, (i) we first evaluate the proposed locally sparse encoding in terms of performance and energy efficiency, and its impact on the resiliency of HDC. (ii) Then, we evaluate the effectiveness of decoding on normal (non-private) HDC in terms of root-mean-square error (RMSE), which is a representative of the quality of data reconstruction. We also report the execution time of the decoding per input sample. (iii) Afterwards, we quantify the proposed private inference by showing how obfuscating of vectors affects the RMSE of the reconstructed data and the inference accuracy. (iv) Finally, we evaluate the proposed private HDC training (both one-pass and iterative), including Privé-HDnn, in terms of accuracy and privacy metric (ε). We also measure the training time of Privé-HDnn and compare with DNN alternatives.

To evaluate the algorithms in terms of accuracy, decoding RMSE, and error resiliency, we used Python 3.9 running on a Linux machine with an Intel Core i7-12700 CPU. To evaluate the efficiency of the proposed locally sparse encoding, we accelerated it on a Xilinx Kintex-7 KC705 FPGA Kit, where we estimated the power using Xilinx Power Estimator (XPE) [64]. For CPU, we estimate the power consumption using CPU Energy Meter [142]. Finally, for Privé-HDnn, in addition to CPU implementation, we use a CUDA implementation on NVIDIA's GTX 1080 Ti GPU for a fair comparison of its training time with previous DNN works as they mainly use GPUs.

(2) Benchmarks: We use standard IoT domain benchmarks: `FACE` detection with face versus non-face labels [86], `UCIHAR` user activity recognition using smartphone (five activities such as sitting, standing, etc.) [143], `PAMAP2` physical activity monitoring using inertial measurement and heart rate monitor (12 activities such as cycling, running, etc.) [144], and `ISOLET` voice of English alphabet recognition [106]. We use `MNIST` [110] and `CIFAR-10` [145] images to evaluate Privé-HDnn. We would like to remind that HDnn achieves the same or better performance compared to DNNs [137] in non-private setting. However, private training is challenging (e.g., previous DNN studies show low accuracies, e.g. 72% on `CIFAR-10` [141]), and applications deal with practical use-cases with a limited number of classes (as opposed to general learning problems). Thus, we compare using `MNIST` [110] and `CIFAR-10` [145] as well.

5.6.1 Encoding

(1) FPGA performance: Fig. 5.6 compares the encoding performance of the approximate locally sparse encoding implemented on FPGA versus normal (non-sparse) encoding on FPGA and CPU. We use $D = 4K$ and set $n = 2m$ (n determines the matrix sparsity; see Fig. 5.4(b)). Note that the performance of non-sparse models is constant for different sparsity levels. We chose `UCIHAR` and `MNIST` as these benchmarks consist of, respectively, the least and highest number of features.

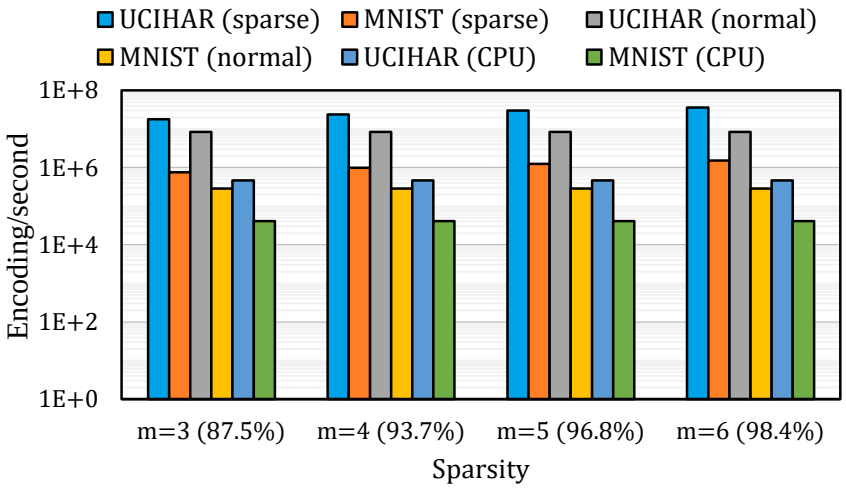


Figure 5.6: Comparing the encoding performance of the approximate locally sparse encoding (FPGA) versus normal encoding on FPGA and CPU.

The sparse FPGA implementation of MNIST, achieves 2.6× (5.3×) higher performance compared to non-sparse FPGA with $m=3$ ($m=6$) sparsity level. These values are slightly lower for UCIHAR, i.e., 2.1× ($m=3$) to 4.3× ($m=6$) because UCIHAR consists of fewer features, so the control overhead slightly lowers its savings.

Compared to CPU implementation, our sparse FPGA implementation achieves 38× ($m=3$) to 79× ($m=6$) higher performance for UCIHAR, and 19× ($m=3$) to 37× ($m=6$) for MNIST. The relative FPGA improvements of UCIHAR is higher because it has less features, so the impact of data movement in CPU becomes more dominant.

(2) FPGA energy: Fig. 5.7 compares the energy consumption of the proposed approximate sparse encoding (FPGA) with the non-sparse FPGA and CPU encoding. Since the power consumption of the sparse and normal FPGA implementation is similar (8 W for UCIHAR and 10 W for MNIST), the energy saving of sparse implementation over the non-sparse FPGA is similar to the performance numbers, e.g., 2.6× (5.3×) with $m=3$ ($m=6$) sparsity level for MNIST. CPU consumes 60 W for UCIHAR and 85 W for MNIST. Accordingly, the proposed sparse FPGA implementation achieves 289× ($m=3$) to 590× ($m=6$) higher performance for UCIHAR, and 157× ($m=3$) to 315× ($m=6$) for MNIST.

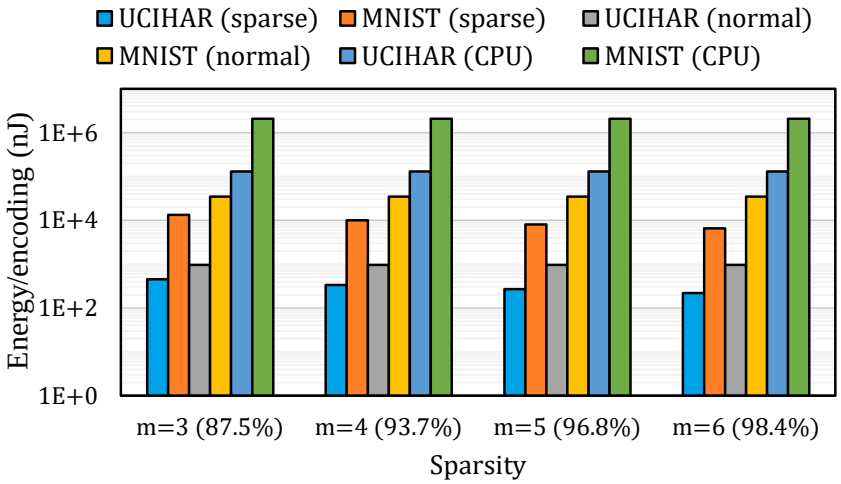


Figure 5.7: Comparing the energy consumption of the approximate locally sparse encoding (FPGA) versus normal encoding on FPGA and CPU.

Table 5.1: Communication bit reduction.

Encoding	m = 3 (87.5%)	m = 4 (93.7%)	m = 5 (96.8%)	m = 6 (98.4%)
Local Sparsity	63%	75%	84%	91%
Top-k	0%	25%	63%	81%

Note that we did not consider the similarity checking step as we assume it is run on the cloud/gateway. Nonetheless, for similarity checking we can expect an improvement proportional to the sparsity level, e.g., 87% for $m=3$ as the number of memory reads and products of class components reduce by 2^m.

(3) Communication efficiency: Data communication energy saving depends on the transmission distance [146]. However, since the transfer energy is linear with the number of bits, we can estimate the transmission energy according to $1-\frac{m}{2^m}$ because per 2^m encoded bits, only $\log 2^m = m$ (non-zero index) is transmitted.

Table 5.1 reports the communication bit reduction (proportional to transmit energy saving) of the proposed local sparsity and compares it with top-k (which needs $\log D$ bits per each non-zero component, where $D=4000$ in this setting). For $m=3$ (87% sparsity), the transmit energy reduces by 62% (compared to 0% of top-k), up to 91% for $m=6$. Note that with $m=3$,

representing the encoded vector of top-k method with its index information increases the number of bits compared to simple binary representation. Thus, for $m = 3$, top-k method transmits the information in the original format, so its compression rate is 0%.

(4) Error resiliency. In normal HDC encoding, the number of encoding zeros and ones are, on expectation, equal. In sparse encoding, however, the majority of bits are zero; hence, a bit-flip of encoded components from zero to one and vice versa has a higher impact on the similarity (dot-product) result. Fig. 5.8 compares the impact of bit error on the accuracy of models using the conventional non-sparse and locally sparse encodings. We model the bit error on the final encoded vectors, so the result is independent of the underlying hardware. The error can be due to hardware error such as emerging memory cells [147] or communication bit error. We adopt the communication error model of [128] which adjusts the parameters of WiFi protocol stack (802.11n), and changes the distance of the transmitter and receiver distance and collects SNR data with Friis propagation loss model.

We show the results on `FACE` and `ISOLET` datasets that have the smallest and largest number of classes, respectively. From Fig. 5.8, we can see that while normal encoding exhibits slightly better robustness, sparse encoding also demonstrates remarkable error robustness. With 2% bit error rate (equivalent to 180 meter-distant transfer), the average accuracy loss of sparse encoding is 0.77% (versus 0.30% of normal encoding). With $\sim$6.5% bit error (250 meter distance), the accuracy loss is 2.37% (versus 1.32% of normal encoding). At these error rates, the accuracy of conventional ML algorithms such as logistic regression and support vector machine drops to zero (random) [128].

5.6.2 Decoding

(1) RMSE: For illustration purpose, Fig. 5.9 shows decoding examples of a handwritten digit and a sinusoidal wave for different encoding techniques. It can be seen that both the decoded image and wave are similar to the original data. To quantify the similarity, Fig. 5.10 shows

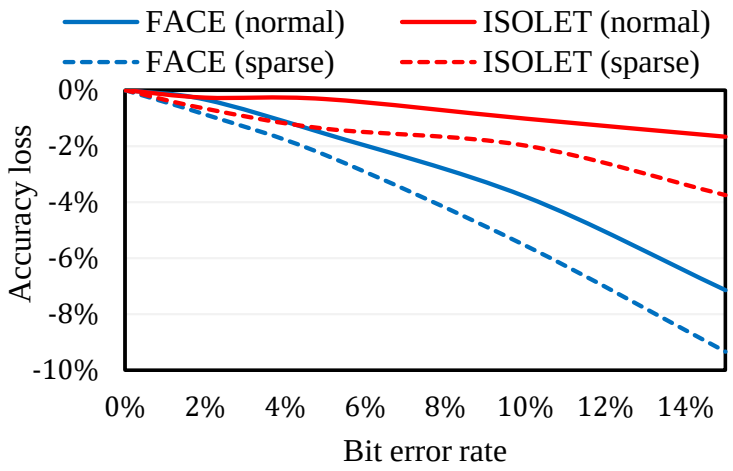


Figure 5.8: Impact of bit error on the accuracy of conventional and sparse encoding. $D = 4000$ for both datasets.

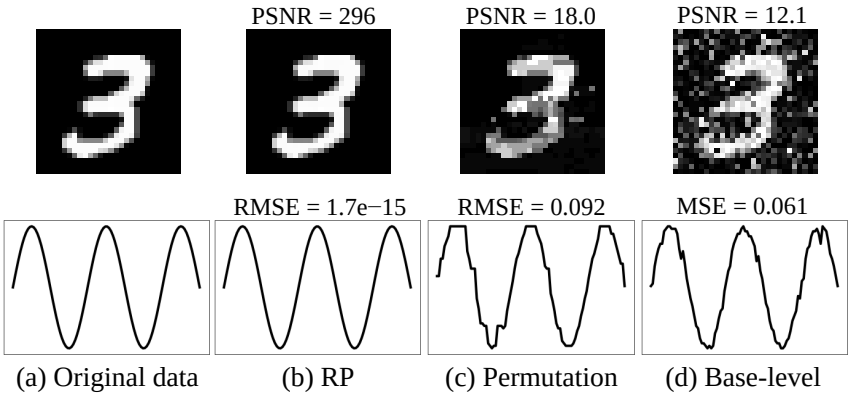


Figure 5.9: Examples of decoded data from handwritten digit and a sinusoidal wave for different encodings.

the RMSE between the original and the decoded data for all our proposed encoding methods (RP, base-level, permutation) and compares it with the state of the art RP [3]. The datasets are normalized to [0, 1] to make the RMSEs more insightful and comparable. Our method of decoding random projection (RP) perfectly reconstructs the original data and achieves an average RMSE of ~ 0 among all benchmarks, while [3] uses an analytical approach to decode the RP and achieves an average RMSE of 0.09. PRID [130] also decodes RP encoding and reports a PSNR of $\sim$51 dB for `MNIST`, whereas our approach achieves an average PSNR of $\sim$250 dB.

The average RMSE of decoding the permutation and base-level is relatively higher, i.e.,

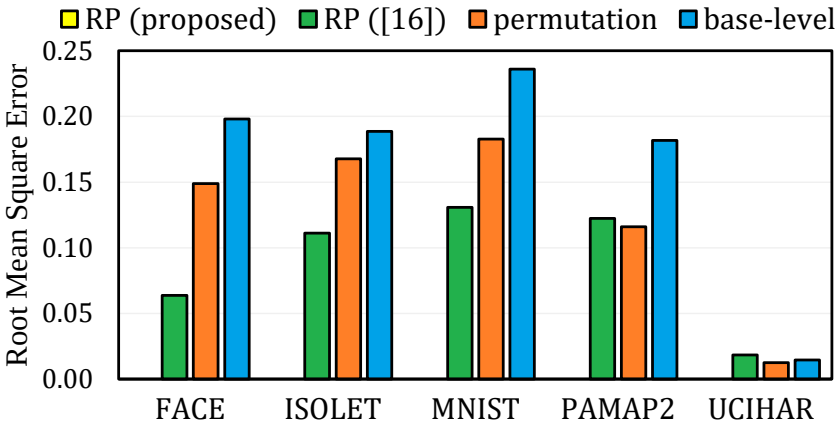


Figure 5.10: RMSE between the original and the decoded data for different encodings (RP, base-level, permutation) along with comparison with analytical RP decoding of [3].

Table 5.2: Execution time (ms) for decoding

Benchmark	RP	Permutation	Base-Level
FACE	0.015	0.86	1.58
ISOLET	0.015	1.14	1.49
MNIST	0.019	1.32	1.99
PAMAP2	0.013	1.06	1.38
UCIHAR	0.002	0.09	1.01

0.126 and 0.164 respectively. The higher RMSE of these methods compared to RP decoding is mainly due to the noise term in Equation (5.3) that affects finding the right $\vec{\mathcal{L}}_x$ that achieves the maximum score. However, even with an RMSE of 0.23 for MNIST base-level decoding, which is the highest RMSE among benchmarks, the constructed image is still recognizable. The noise impact, hence the decoding quality, is directly related to the number of features (d in Equation (5.3)). UCIHAR has the least number of features per sample ($d = 27$), so its decoding achieves a small RMSE of < 0.02. The other datasets have 561 to 784 features, and thus they have larger and similar RMSEs.

(2) Execution time: Table 5.2 reports the execution time in milliseconds for decoding each input sample. To decode each feature of a sample, RP uses D multiplications and each input can be decoded in 0.013 mSec on average. The permutation and base-level encoding use $D \times \mathcal{L}$

Table 5.3: Accuracy and decoding RMSE for sparse encoding.

Sparsity	FACE (95%)	UCIHAR (95%)	PAMAP2 (94.5%)	ISOLET (94%)	MNIST (95.5%)
87.5% (m = 3)	-0.3%	0.0%	-0.1%	-0.2%	-0.1%
93.7% (m = 4)	-1.1%	-1.6%	-1.7%	-1.3%	-2.2%
96.8% (m = 5)	-2%	-2.9%	-5.5%	-3.0%	-7%
98.4% (m = 6)	-3.3%	-20.0%	-10.0%	-5.8%	-9.5%
RMSE	0.173–0.184	0.282–0.346	0.72–0.735	0.616–0.624	0.316–0.332

(for $\mathcal{L}$ being the number of levels) multiplications, yet decoding of these methods is also fast (0.90 mSec for permutation and 1.49 mSec for base-level).

5.6.3 Inference Privacy

RMSE and Accuracy: In the previous subsection we observed that RP encoding exhibits the highest exposure of data. Thus, for brevity, we evaluate the impact of the proposed local sparsification technique (Section 5.4) in obfuscating the RP encoding. Table 5.3 summarizes the accuracy of benchmarks for different local sparsity rates, where, e.g., $m=4$ means only one dimension out of $2^4=16$ consecutive dimensions remains one, so the sparsity rate is $1-\frac{1}{16}=87.5\%$. The first row of the table reports the baseline non-sparse accuracy (e.g., 95.0% for `FACE`). The average accuracy loss of $m=3$ ($m=4$) is only 0.14% (1.6%) among all benchmarks.

The last row of the table reports the RMSE *range* of the decoded benchmarks. The range corresponds to $m=3$ (smallest RMSE) to $m=6$ (largest RMSE). To recap, without sparsification, the RMSEs of all decoded benchmarks are ~ 0 as discussed in subsection 5.6.2. Based on Table 5.3, with $m=3$, with an accuracy degradation of only 0.14%, increases the RMSE to 0.421, meaning that, on average, the retrieved features are deviated by 0.421 from the original values (features are between 0 and 1). This is notable especially when the features are independent measurements such as a vector of temperature, heart rate, and inertial measurements (as in `PAMAP2`), so the exact values cannot be recovered anymore.

For illustration purpose, Fig. 5.11 shows how samples of handwritten digit and sinusoidal

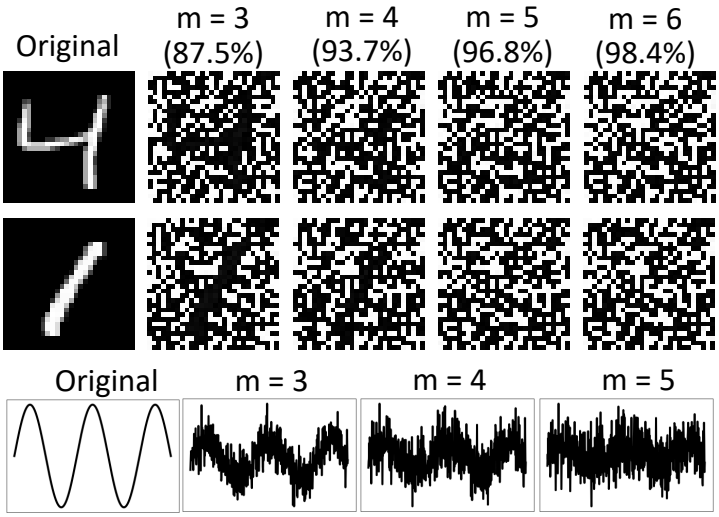


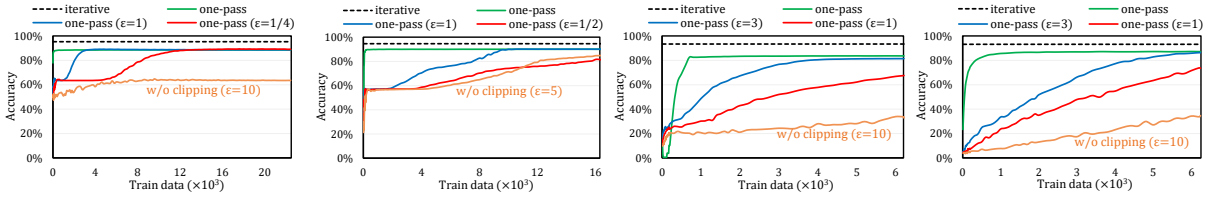
Figure 5.11: Decoding the sparse vectors of RP encoding.

wave are perturbed by locally sparse encoding. The average PSNR among all the inputs drops below 12 dB, and RMSE of the sinusoidal wave increases to 0.7 (versus $\sim$0 of original RP).

5.6.4 One-pass Training Privacy

Fig. 5.12 shows the results of private one-pass training. We set δ=10^{-5} ($< \frac{1}{|\mathcal{D}|}$ for all datasets) and used D=4K dimensions. At each number of inputs (consumed train data on the x-axis), we obtained the accuracy using the entire test set for inference. Each curve is the average of 20 runs. The dashed curve (labeled *iterative*) shows the accuracy of non-private iterative HDC training. The green curve (labeled *one-pass*) shows the typical non-private online (one-pass) training which achieves 87.4% accuracy on average ($-$6.8% versus iterative training). The sharp spike of one-pass learning indicates the fast learning of HDC from limited data, e.g., in `FACE` detection and `UCIHAR` one-pass learning reaches near 0.5% of its maximum accuracy by learning from only 2% of the data.

The private one-pass training of `FACE` and `UCIHAR` benchmarks yields the same accuracy of non-private one-pass with $\varepsilon = 1$ which is a tight privacy constraint. `FACE` can obtain the baseline one-shot accuracy with $\varepsilon = \frac{1}{4}$ as well, while `UCIHAR` could not converge with $\varepsilon = \frac{1}{2}$. This can be



(a) FACE detection (b) UCIHAR activity recognition (c) PAMAP2 physical activity recog. (d) ISOLET speech recognition

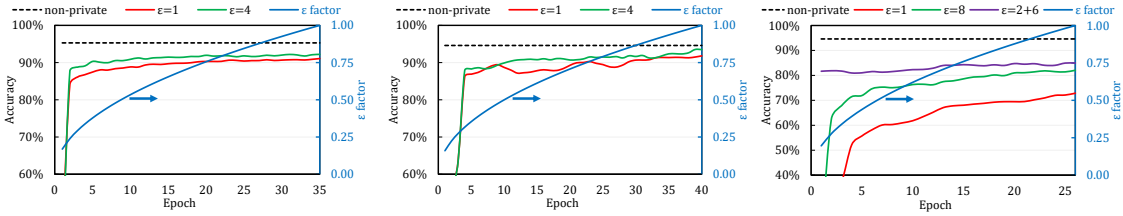
Figure 5.12: Accuracy of differentially private one-pass HDC training ($\delta = 10^{-5}$, $D = 4000$).

attributed to the fact that FACE has more data and fewer labels (two labels versus five of UCIHAR) which makes it more resilient to noise. The magnitude of noise for one-pass learning depends on the sensitivity, $\|\vec{\mathcal{H}}\|_2$, which is independent of the dataset size. Since size of the dataset affects the value of class components, with more vector accumulation in datasets with more samples, the $\vec{C}_i \cdot \vec{\mathcal{H}}$ score in Equation (5.7) grows versus the noise term $\vec{\mathcal{N}} \cdot \vec{\mathcal{H}}$. The other two benchmarks, PAMAP2 and ISOLET, have more classes (12 and 26) with less training data. As a result, they could converge to their maximum accuracy at larger $\varepsilon = 3$ (i.e., less noise).

In Section 5.5.2, we proposed to normalize and clip the vectors to reduce the HDC sensitivity, and as a result, the amount of noise. The "*w/o clipping*" curves of Fig. 5.12 show the accuracy of private learning without normalization and clipping. Accordingly, due to the large amount of noise entailed, only weak privacy ($\varepsilon = 10$) could be achieved, with 32.7% less accuracy (versus one-pass private training that benefits from normalization and clipping).

5.6.5 Iterative Training Privacy

Fig. 5.13 compares the accuracy of the benchmarks trained with differentially private iterative HDC training. With $\varepsilon = 1$ ($\varepsilon = 4$), private FACE benchmark achieves 91.1% (92.2%) accuracy, which is 2.5% (3.6%) higher than one-pass training. Similarly, private UCIHAR achieves 1.8% (3.5%) better accuracy than one-pass. Notably, with $\varepsilon = 4$, differentially private UCIHAR's accuracy is only 1% lower than the non-private training. PAMAP2, however, has more classes and less training data. Thus, even with $\varepsilon = 8$, it achieves 82% accuracy, which is not any better



(a) FACE detection (b) UCIHAR activity recognition (c) PAMAP2 physical activity recognition

Figure 5.13: Accuracy of differentially private iterative HDC training ($\delta = 10^{-5}$, $D = 4{,}000$).

than one-pass private training. This is because of the same reason we also observed in one-pass training, i.e., due to the small training set of PAMAP2, the noise impact ($\vec{\mathcal{N}} \cdot \vec{\mathcal{H}}$) aggravates versus the similarity score $\vec{C}_i \cdot \vec{\mathcal{H}}$. Particularly, because of larger number of classes, the margin between classes is also smaller. Thus, for PAMAP2, we first trained a one-pass private model with $\varepsilon = 2$ for faster initial convergence, followed by an iterative training with $\varepsilon = 6$. The resultant model $(2+6)$ obtains 3% higher accuracy compared to the $\varepsilon = 8$ iterative training from scratch.

For a target ε and δ, various combinations of σ, q, and T can satisfy Equation (5.9). We observed that smaller noise parameter σ achieves better accuracy at the cost of fewer epochs. Accordingly, we set $T = 16 \times 10^4$ and $q = \frac{0.1}{\sqrt{T}}$ which results in a unique and small σ_{min} to satisfy a (ε, δ). Also, as the training proceeds, the number of epochs $E = q \times T$ is the point the privacy target ε is achieved. The $'\varepsilon$ factor$'$ in Fig. 5.13 (the secondary Y axis) shows the value of ε at each epoch. Before reaching the last epoch, achieved privacy is tighter, though the accuracy is also lower. For instance, in Fig. 5.13a, in the 10^{th} epoch, the ε factor is $\simeq 0.5$ which means the green curve setting (final $\varepsilon = 4$) has a better ε of 2 if we terminate the training there.

5.6.6 Image Classification

(1) Accuracy: Fig. 5.14 shows the baseline non-private HDnn and Privé-HDnn accuracy on MNIST and CIFAR-10 datasets. The non-private HDnn achieves 97.8% and 87.0% accuracy on MNIST and CIFAR-10. Note that the baseline HDnn here uses a ResNet-18 network pretrained

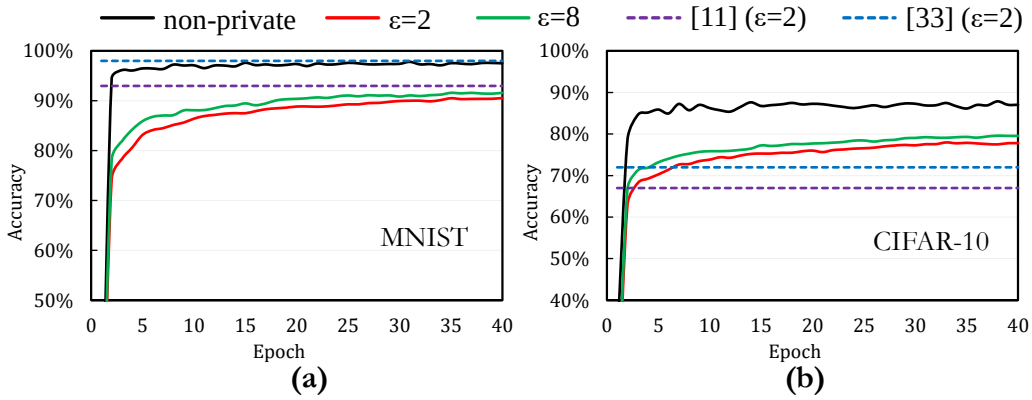


Figure 5.14: Private training on images with Privé-HDnn.

on ILSVRC 2012 dataset as the feature extractor, *without* tuning the CNN on the target `MNIST` and `CIFAR-10`, which results in accuracy drop compared to tuned HDnn (that otherwise achieves comparable or better accuracy than DNN [137]) or baseline ResNet-18. With $\varepsilon = 2$, Privé-HDnn achieves 90.6% on `MNIST`, and 77.8% on `CIFAR-10`. By relaxing the ε to 8, the accuracy improves to 91.6% on `MNIST`, and 79.5% on `CIFAR-10`.

The accuracy numbers for differentially-private `MNIST` DNNs range from 90% in [148] to 98.1% (with $\varepsilon = 2.93$) in [141], where the latter work creates new models from scratch with minimal parameters to make the DNN more noise tolerant. For `CIFAR-10`, the study in [141] reports a maximum accuracy of 72% with $\varepsilon = 2$, which is 5% higher than the pioneer work of Abadi *et al.* [43]. Privé-HDnn could achieve 77.8% on the same `CIFAR-10` dataset which is 5.8% higher than [141], with the same $\varepsilon = 2$. Thus, the accuracy of Privé-HDnn is comparable or better than previous deep learning studies with differential privacy. The accuracy of Privé-HDnn can be further improved by using larger CNN backbone such as ResNet-50 or using more relevant public dataset (e.g., a more relevant subset of ILSVRC 2012).

(2) Execution time: Unlike studies like [141] that build a custom model, Privé-HDnn leverages existing models for feature extraction and only trains HDC on the extracted features. Due to the fast convergence of HDC and its simplicity, Privé-HDnn needs only 40 training epochs (Fig. 5.14) and each epoch takes $\sim$2.0 second on GPU, and $\sim$8.7 second on CPU. Table 5.4

Table 5.4: Training time (minutes) of Privé-HDnn versus previous studies.

Dataset	**Privé-HDnn GPU**	**Privé-HDnn CPU**	[43] (low)	[43] (high)	[141] (low)	[141] (high)
MNIST	1.3 (1.0×)	5.8 (4.5×)	12 (9.2×)	533 (410×)	28 (22×)	160 (123×)
CIFAR-10	1.3 (1.0×)	5.8 (4.5×)	40 (31×)	467 (359×)	280 (215×)	1600 (1231×)

compares the training time of Privé-HDnn with DNN training in [43] and [141]. The *low* and *high* indicates the minimal and maximum configuration of the previous works based on their target privacy budgets and hyperparameters (e.g., [141] *high* configuration uses 400 epochs with batch size of 256 and consumes 240 second per epoch for the CIFAR-10 dataset). The GPU implementation of Privé-HDnn improves the runtime by 9.2–410× over [43], and by 22–1231× over [141]. Even the CPU training of Privé-HDnn takes less time than previous DNN works that run on GPU. The CPU training time is 2.1–92× smaller than [43], and 4.8–276× smaller than [141].

5.6.7 Overhead

The proposed privacy-enhancing techniques impose minimal overhead. The accuracy drop of the local sparsification is only 0.14% for 87% sparsification rate. This rate of sparsification lowers the energy cost of communicating data by 63% and improves performance by 2.6× (38×) over non-sparse FPGA (CPU). The noise is applied on top of the model, so the model size and training time are kept intact. In particular, for images, Privé-HDnn leverages the readily available pre-trained CNNs (unlike previous studies that train from scratch), so only HDC is trained, which is very fast and more accurate; 1.3 minutes compared to 26 hours for the CNN.

5.7 Conclusion

In this chapter, we leveraged the noise robustness of HDC for efficient privacy-preserved inference and training. To improve inference privacy, we proposed local sparsification with

efficient FPGA implementation, which deviates the decoded values by a normalized RMSE of 0.42 versus the original values, and improves the encoding performance by up to 5.3× over non-sparse FPGA implementation, and 79× over CPU implementation. The encoding energy consumption is also reduced by 5.3× compared with non-sparse FPGA implementation, and 590× versus CPU implementation. The communication energy (bit transfer) also reduces by 63%–91%. For training, we proposed differentially-private one-pass and iterative training. Thanks to low sensitivity of one-pass training and noise robustness of HDC, one-pass private training retained the same accuracy of non-private models while achieving stringent privacy. With private iterative training, the average accuracy of the models further improved by 3.4%. Finally, we used a hybrid CNN-HDC, Privé-HDnn, for differentially-private training on images, where the CNN feature extractor is fixed and privacy preserving training only happens on the HDC part. Privé-HDnn achieves comparable or better accuracy by up to 5.8% versus DNN-only solutions, while reducing the training time on GPU 9.2–1231× over the state of the art [43] [141].

The next chapter summarizes the proposed approaches on bringing efficient computing and learning to the edge and discloses the future directions.

Chapter 5, in full, is currently being prepared for submission for publication of the material "Private Learning and Inference with Hyperdimensional Computing" by Behnam Khaleghi, Jaeyoung Kang, Xiaofan Yu, and Tajana Rosing, to be submitted to IEEE Internet of Things Journal. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Summary and Future Work

The growing number of interconnected devices and systems produce an unprecedented volume of data. Notably, the growing number of edge devices in various shapes continually produces data that need immediate processing with sustainable energy consumption. With the current processing systems being pushed to their limits, striking methods to save energy is crucial. In this dissertation, we propose two classes of solutions. First, we target the energy efficiency of FPGAs. Relatively higher efficiency of these devices compared to CPUs and GPUs, together with their programmability are making FPGAs a fundamental computation unit of both data centers and edge computing applications. In Chapter 2 we present more aggressive power and energy reduction of designs mapped onto FPGAs. Our approach has a broad impact as it can be implemented on FPGAs running general applications from data filtering to heavy signal processing at different levels of the computing stack.

In the rest of the dissertation, we target more energy efficient alternatives of todays mainstream machine learning. The large model size and computation demand of ML algorithms, particularly DNNs, hinder deploying them on the majority of edge devices. Real-time response requirements, together with network reliability and congestion issues, impede using cloud for

remote learning services. To cope with these challenge, we bring intelligence to the edge by using Hyperdimensional Computing (HDC) as a lightweight alternative to DNNs. We first propose an efficient FPGA implementation of HDC with approximate computing mode that can even further lower the resource requirements of HDC and facilitate its implementation on very low-end FPGAs in Chapter 3. Then, to enable HDC-based training for ultra-low-energy wearable and sensory devices, we propose a flexible ASIC design that supports training, inference, and clustering on a diverse range of applications n Chapter 4. The last part of this dissertation addresses on the privacy challenges learning using HDC and DNNs. We show that although HDC is not inherently private, its noise robustness can be leveraged for efficient privacy-preserved inference and training by obfuscating the information through noise robustness with insignificant impact on accuracy. To improve inference privacy, we propose approximate sparsification with efficient FPGA implementation, which reduces the amount of transmitted bits, and hence the communication energy as well. We also design differentially-private one-pass and iterative training with tight privacy budget and good accuracy. Finally, for larger image datasets, we design the combination of HDC and CNN that improves the accuracy with three orders of magnitude smaller training time on GPUs.

6.1 Dissertation Summary

Efficiency by Aggressive Energy Reduction in FPGA-based Designs: In Chapter 2, we proposed a systematical approach to leverage the available thermal headroom of FPGA-mapped designs for power and energy improvement. By comprehensively analyzing the timing and power consumption of FPGA building blocks under varying temperatures and voltages, we proposed a thermal-aware voltage scaling flow that effectively utilizes the thermal margin to reduce power consumption without degrading performance. We showed that the proposed flow could be employed for energy optimization as well, whereby power consumption and delay are

compromised to accomplish the tasks with minimum energy. This is desirable for a majority of edge applications with tight energy requirements. Lastly, we proposed a simulation framework to be able to examine the efficiency of the proposed method for other applications that are inherently tolerant to a certain amount of error, granting further power saving opportunity. Experimental results showed up to 36% power reduction with the same performance and 66% total energy saving when energy is the optimization target.

Exact and Approximate FPGA Implementation of HDC Inference: In Chapter 3, we proposed SHEAR*er*, an algorithm-hardware co-optimization to improve the performance and energy consumption of HD computing. We gained insight from a prudent scheme of approximating the hypervectors that, thanks to the error resiliency of HD, has minimal impact on accuracy while providing high prospect for hardware optimization. Unlike previous works that generate the encoding hypervectors in full precision and then perform ex-post quantization, we computed the encoding in an approximate manner that saves resources yet affords high accuracy. We also proposed a novel FPGA architecture that achieves striking performance through massive parallelism with low energy consumption. We also developed a software framework that enables training HD models by emulating the proposed approximate encodings. Such approximate-aware training is crucial to avoid significant accuracy loss. The FPGA implementation of SHEAR*er* achieved an average performance boost of 15.7× and energy savings of up to 301× compared to state-of-the-art encoding methods implemented on GeForce GTX 1080 Ti using machine learning datasets of the IoT domain.

Highly-Efficient ASIC Design of HDC Learning and Inference: In Chapter 4, to realize a custom and tiny HDC engine that can be deployed in wearable and extremely low-energy sensory devices while supporting a diverse range of applications, we proposed a novel encoding that achieves high accuracy for various IoT applications. Thereafter, we leveraged the proposed encoding and designed a highly efficient and flexible ASIC accelerator suited for the edge domain. The proposed design supports both classification (train and inference) and

clustering for unsupervised learning on edge. It is also flexible in terms of input size (hence it can run various applications) and hypervectors dimensionality, allowing it to trade off the accuracy and energy/performance on-demand. The proposed design improved the prediction accuracy over previous HDC and ML techniques by 3.5% and 6.5%, respectively. It occupies an area of 0.30 mm^2 (at 14 nm node), and consumes 0.09 mW static and 1.97 mW active power at 500 MHz, which can be further throttled by simply operating at lower frequencies.

Preserving the Privacy of HDC-based Learning and Inference: In Chapter 5, we used HD computing to cope with the privacy challenges of deep neural networks. We first show that HDC is not inherently privacy-preserved, and in fact, the raw data can be reconstructed from the encoded vectors. Thus, transmitting the encoded data such as in federated learning or cloud inference setting jeopardizes privacy. Superior error tolerance of HDC can be leveraged to enhance the privacy by obfuscating the information through noise/error injection with insignificant impact on accuracy. To improve inference privacy, we proposed local sparsification with approximate FPGA implementation, which deviates the decoded values by a normalized root-mean-square error (RMSE) of 0.42, and reduces the encoding energy consumption by up to 2.1–5.3× (157–590×) over the baseline FPGA (CPU) implementation. We designed differentially-private one-pass and iterative training. We modify the vectors to lower the sensitivity (hence, additive noise) of one-pass training, as a result of which, one-pass HDC could retain the same accuracy of non-private models while obtaining stringent privacy guarantees. With private iterative training, the average accuracy of the models improved by 3.4%. Finally, we used a hybrid CNN-HDC for private training over larger image data, where pretrained CNN is a fixed feature extractor and the HDC classifier is trained with privacy preserving guarnatees. Compared to DNN-only solutions, the hybrid model achieves up to 5.8% higher accuracy and is up to 1231× faster when training on GPU.

6.2 Future Directions

To make HD computing ubiquitous for computing at the edge, we plan to extend the HD computing from both hardware and algorithm perspectives in the following way:

• **Coarse-Grained Reconfigurable Architecture (CGRA):** Both IoT applications and HDC algorithms are continually evolving. For instance, in the case of seizure detection from more channels and/or heterogeneous sources of data (such as in-ear measurements combined with brain signals), we need more flexibility in the hardware. In Chapter 4, to make an ASIC design workable for various types of IoT applications, we studied some of the existing HDC encoding algorithms and proposed a new encoding that achieves high accuracy for a number of different applications. From the proposed GENERIC architecture, we learned that the memory capacity and bandwidth dictate the overall area and power consumption of the ASIC design (which will not vary considerably between an ASIC and CGRA). To avoid the large form-factor and overheads incurred by the programmable logic of FPGAs, we will seek CGRA architecture for HDC to bridge the flexibility and efficiency gap between FPGA and ASIC implementation. It entails exploring and identifying the common atomic units of different HDC algorithms, proposing building blocks and architecture, and domain-specific language and software flow to write and map the HD applications to the CGRA. We believe that a well-designed CGRA architecture can provide comparable energy and performance to ASIC with a flexibility sufficient for HDC algorithms.

• **In-Memory Implementation of HDC:** SRAM memory that stores the class vectors contributes to more than 98% of the static power and area, and 90% of the dynamic power in our proposed HDC GENERIC engine presented in Chapter 4. As a future work, we will seek the opportunities to use dense non-volatile memories such as ReRAM to replace the SRAM memories. ReRAM cells are CMOS compatible, have a footprint of $4F^2$ (versus $140F^2$ of SRAM), and consume significantly less static power. We will investigate the in-memory search capability of ReRAM

crossbars to perform similarity checks with less energy and higher performance. Several studies have used ReRAM-based crossbars for HD computing. These studies, however, do not consider the device-level limitations of in-memory computing, such as limited number of memory rows that can be activated, I-V non-idealities, time- and thermal-induced resistant drift, sense-amplifier and ADC errors, etc. Using the measured characteristics of ReRAM provided by our industry partner (TSMC), we will explore the circuit-level peripherals and algorithms (e.g., how to break down the integer operations into bit-level) to realize in-memory HD computing while accurately taking the aforementioned non-idealities into account. Similar to our work in Chapter 3, we will explore techniques to account the circuit non-idealities during the HDC training in order to minimize their impact.

• **Life-Long Learning with HDC:** Life-long learning enables systems to learn continuously in changing environments without forgetting past lessons, and being able to use the previous knowledge to better learn from the current observations. Life-long learning with DNNs is challenging due to the significant computational resources demanded, that is infeasible for the majority of edge devices. We plan to build a system to facilitate life-long learning with HD computing. Using HDC, we first develop algorithms that can model and track the current context and its changes by density-based clustering. The hardware system uses these algorithmic innovations to enable efficient lifelong learning for distributed sensing. The remote sensing device will store the density model of the current context in a working in-memory block that allows extremely fast access and updates. Using life long learning, the device can update this model once new data is received from the environment. Upon a change in context, detected using the developed algorithms, the device will fetch the most similar model from a persistent storage (e.g., non-volatile flash memory) and load it into the working memory. If no sufficiently similar context is found, the device will create a new context in working memory. All the frequent search operations will rely on efficient in-memory and in-storage search, while the encoding and update calculations will rely upon tiny CGRA embedded in the edge device.

Bibliography

[1] C. Chiasson and V. Betz, "Should fpgas abandon the pass-gate?" in *FPL*, vol. 13, 2013, pp. 1–8.

[2] B. Khaleghi, S. Salamat, A. Thomas, F. Asgarinejad, Y. Kim, and T. Rosing, "Shear er: highly-efficient hyperdimensional computing by software-hardware enabled multifold approximation," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 241–246.

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

[4] "Number of internet of things (iot) connected devices worldwide," https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/, accessed: 2022-10-08.

[5] "Volume of data/information created," https://www.statista.com/statistics/871513/worldwide-data-created/, accessed: 2022-10-08.

[6] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein, "Scaling, power, and the future of cmos," in *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.* IEEE, 2005, pp. 7–pp.

[7] I. L. Markov, "Limits on fundamental limits to computation," *Nature*, vol. 512, no. 7513, pp. 147–154, 2014.

[8] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*. IEEE, 2016, pp. 20–26.

[9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[10] J. Shalf, "The future of computing beyond moore's law," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190061, 2020.

[11] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–4.

[12] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 5–14.

[13] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 13–24, 2014.

[14] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 1–14.

[15] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung *et al.*, "Azure accelerated networking:{SmartNICs} in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 51–66.

[16] D. Blaauw, D. Sylvester, P. Dutta, Y. Lee, I. Lee, S. Bang, Y. Kim, G. Kim, P. Pannuto, Y.-S. Kuo *et al.*, "Iot design space challenges: Circuits and systems," in *2014 Symposium on VLSI technology (VLSI-technology): digest of technical papers*. IEEE, 2014, pp. 1–2.

[17] S. Venkataramani, K. Roy, and A. Raghunathan, "Efficient embedded learning for iot devices," in *Design Automation Conference (ASP-DAC), 21st Asia and South Pacific*. IEEE, 2016, pp. 308–311.

[18] M. Elnawawy, A. Farhan, A. Al Nabulsi, A.-R. Al-Ali, and A. Sagahyroon, "Role of fpga in internet of things applications," in *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE, 2019, pp. 1–6.

[19] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, 2006, pp. 21–30.

[20] A. Boutros, S. Yazdanshenas, and V. Betz, "You cannot improve what you do not measure: Fpga vs. asic efficiency gaps for convolutional neural network inference," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[21] U. S. Shanthamallu, A. Spanias, C. Tepedelenlioglu, and M. Stanley, "A brief survey of machine learning methods and their sensor and iot applications," in *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 2017, pp. 1–8.

[22] Z. Zhou, H. Liao, B. Gu, K. M. S. Huq, S. Mumtaz, and J. Rodriguez, "Robust mobile crowd sensing: When deep learning meets edge computing," *IEEE Network*, vol. 32, no. 4, pp. 54–60, 2018.

[23] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[24] R. Aitken, V. Chandra, J. Myers, B. Sandhu, L. Shifren, and G. Yeric, "Device and technology implications of the internet of things," in *2014 symposium on VLSI technology (VLSI-technology): digest of technical papers*. IEEE, 2014, pp. 1–4.

[25] G. Tzimpragos, A. Madhavan, D. Vasudevan, D. Strukov, and T. Sherwood, "Boosted race trees for low energy classification," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 215–228.

[26] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 199–213.

[27] X. Yu, X. Song, L. Cherkasova, and T. Š. Rosing, "Reliability-driven deployment in energy-harvesting sensor networks," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–9.

[28] I. Ahmed, S. Zhao, J. Meijers, O. Trescases, and V. Betz, "Automatic bram testing for robust dynamic voltage scaling for fpgas," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 68–687.

[29] L. L. Shen, I. Ahmed, and V. Betz, "Fast voltage transients on fpgas: Impact and mitigation strategies," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 271–279.

[30] 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.

[31] A. Thomas, S. Dasgupta, and T. Rosing, "Theoretical foundations of hyperdimensional computing," *Journal of Artificial Intelligence Research*, vol. 72, pp. 215–249, 2021.

[32] 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.

[33] A. Thomas, B. Khaleghi, G. K. Jha, N. Himayat, R. Iyer, N. Jain, and T. Rosing, "Streaming encoding algorithms for scalable hyperdimensional computing," *arXiv preprint arXiv:2209.09868*, 2022.

[34] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations," *ACM Computing Surveys (CSUR)*.

[35] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahim, "A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges," *arXiv preprint arXiv:2112.15424*, 2021.

[36] A. Rahimi, P. Kanerva *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *International Symposium on Low Power Electronics and Design*, 2016, pp. 64–69.

[37] A. Moin, A. Zhou, A. Rahimi, A. Menon, S. Benatti, G. Alexandrov, S. Tamakloe *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.

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

[39] M. Eggimann, A. Rahimi, and L. Benini, "A 5μ w standard cell memory-based configurable hyperdimensional computing accelerator for always-on smart sensing," *arXiv preprint arXiv:2102.02758*, 2021.

[40] S. Datta *et al.*, "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.

[41] R. Chandrasekaran, K. Ergun, J. Lee, D. Nanjunda, J. Kang, and T. Rosing, "Fhdnn: communication efficient and robust federated learning for aiot networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 37–42.

[42] 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)*. IEEE, 2019, pp. 435–446.

[43] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[44] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," *arXiv preprint arXiv:1602.04283*, 2016.

[45] "Intel stratix 10 logic array blocks and adaptive logic modules user guide," User Guide, Intel, September 2017.

[46] "Lowering power at 28 nm with xilinx 7 series devices," White Paper, Xilinx, January 2015.

[47] M. Bao, A. Andrei, P. Eles, and Z. Peng, "On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration," in *Design Automation Conference, 46th ACM/IEEE*. IEEE, 2009, pp. 490–495.

[48] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter, "Active management of timing guardband to save energy in power7," in *Microarchitecture (MICRO), 44th Annual IEEE/ACM International Symposium on*. IEEE, 2011, pp. 1–11.

[49] H. Amrouch, B. Khaleghi, and J. Henkel, "Voltage adaptation under temperature variation," in *15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. IEEE, 2018, pp. 57–60.

[50] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee *et al.*, "The stratix ii logic and routing architecture," in *Proceedings of the ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. ACM, 2005, pp. 14–20.

[51] Z. Seifoori, Z. Ebrahimi, B. Khaleghi, and H. Asadi, "Introduction to emerging sram-based fpga architectures in dark silicon era," *Advances in Computers*, 2018.

[52] J. M. Levine, E. Stott, and P. Y. Cheung, "Dynamic voltage & frequency scaling with online slack measurement," in *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*. ACM, 2014, pp. 65–74.

[53] J. Nunez-Yanez, "Adaptive voltage scaling in a heterogeneous fpga device with memory and logic in-situ detectors," *Microprocessors and Microsystems*, vol. 51, pp. 227–238, 2017.

[54] S. Zhao, I. Ahmed, C. Lamoureux, A. Lotfi, V. Betz, and O. Trescases, "Robust self-calibrated dynamic voltage scaling in fpgas with thermal and ir-drop compensation," *IEEE Transactions on Power Electronics*, vol. 33, no. 10, pp. 8500–8511, 2018.

[55] A. Amouri, H. Amrouch, T. Ebi, J. Henkel, and M. Tahoori, "Accurate thermal-profile estimation and validation for fpga-mapped circuits," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*. IEEE, 2013, pp. 57–60.

[56] B. Salami, O. S. Unsal, and A. Cristal Kestelman, "Comprehensive evaluation of supply voltage underscaling in fpga on-chip memories," in *Proceedings of the 51th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2018.

[57] B. Khaleghi, B. Omidi, H. Amrouch, J. Henkel, and H. Asadi, "Estimating and mitigating aging effects in routing network of fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 3, pp. 651–664, 2019.

[58] C. Chiasson and V. Betz, "Coffe: Fully-automated transistor sizing for fpgas," in *Field-Programmable Technology (FPT), 2013 International Conference on*. IEEE, 2013, pp. 34–41.

[59] Predictive technology model. [Online]. Available: http://ptm.asu.edu/

[60] S. Yazdanshenas, K. Tatsumura, and V. Betz, "Don't forget the memory: Automatic block ram modelling, optimization, and architecture exploration," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 115–124.

[61] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger, "A 90nm low-power fpga for battery-powered applications," in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*. ACM, 2006, pp. 3–11.

[62] "Stratix iv device handbook," Datasheet, Intel, September 2014.

[63] Nangate open cell library. [Online]. Available: http://nangate.com/

[64] "Xilinx power estimator user guide," User Guide, Xilinx, 2018.

[65] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed *et al.*, "Vtr 7.0: Next generation architecture and cad system for fpgas," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, p. 6, 2014.

[66] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan, "Architectural enhancements in stratix-iii™ and stratix-iv™," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2009, pp. 33–42.

[67] J. Lamoureux and S. J. Wilton, "Activity estimation for field-programmable gate arrays," in *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*. IEEE, 2006, pp. 1–8.

[68] R. Zhang, M. R. Stan, and K. Skadron, "Hotspot 6.0: Validation, acceleration and extension," *University of Virginia, Tech. Rep*, 2015.

[69] S. Velusamy, W. Huang, J. Lach, M. Stan, and K. Skadron, "Monitoring temperature in fpga based socs," in *2005 International Conference on Computer Design*. IEEE, 2005, pp. 634–637.

[70] "Powerplay early power estimator user guide," User Guide, Intel, February 2017.

[71] "Timing closure user guide," User Guide, Xilinx, January 2012.

[72] H. Amrouch, B. Khaleghi, and J. Henkel, "Optimizing temperature guardbands," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 175–180.

[73] C. Beckhoff, D. Koch, and J. Torresen, "The xilinx design language (xdl): Tutorial and use cases," in *6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, 2011, pp. 1–8.

[74] "Intel fpga temperature sensor ip core user guide," User Guide, Intel, May 2018.

[75] E. A. Burton, G. Schrom, F. Paillet, J. Douglas, W. J. Lambert, K. Radhakrishnan, and M. J. Hill, "Fivr—fully integrated voltage regulators on 4th generation intel® core™ socs," in *2014 IEEE Applied Power Electronics Conference and Exposition-APEC 2014*. IEEE, 2014, pp. 432–439.

[76] S. Tian and J. Szefer, "Temporal thermal covert channels in cloud fpgas," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019, pp. 298–303.

[77] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2011, pp. 189–196.

[78] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "Reliability-aware design to suppress aging," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[79] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 19.

[80] 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)*. IEEE, 2017, pp. 445–456.

[81] P. A. Jamieson and K. B. Kent, "Odin ii: an open-source verilog hdl synthesis tool for fpga cad flows," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2010, pp. 288–288.

[82] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.

[83] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[84] J. J. Zhang and S. Garg, "Fate: fast and accurate timing error prediction framework for low power dnn accelerator design," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2018, p. 24.

[85] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *arXiv preprint arXiv:1807.08583*, 2018.

[86] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[87] A. T. Elthakeb, P. Pilligundla, A. Yazdanbakhsh, F. Mireshghallah, and H. Esmaeilzadeh, "Releq: A reinforcement learning approach for deep quantization of neural networks," *arXiv preprint arXiv:1811.01704*, 2018.

[88] N. Y. Masse, G. C. Turner, and G. S. Jefferis, "Olfactory information processing in drosophila," *Current Biology*, vol. 19, no. 16, pp. R700–R713, 2009.

[89] G. C. Turner, M. Bazhenov, and G. Laurent, "Olfactory representations by drosophila mushroom body neurons," *Journal of Neurophysiology*, vol. 99, no. 2, pp. 734–746, 2008.

[90] R. I. Wilson, "Early olfactory processing in drosophila: mechanisms and principles," *Annual Review of Neuroscience*, vol. 36, pp. 217–241, 2013.

[91] B. A. Olshausen and D. J. Field, "Sparse coding of sensory inputs," *Current Opinion in Neurobiology*, vol. 14, no. 4, pp. 481–487, 2004.

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

[93] M. Wan, A. Jönsson, C. Wang, L. Li, and Y. Yang, "Web user clustering and web prefetching using random indexing with weight functions," *Knowledge and information systems*, vol. 33, no. 1, pp. 89–115, 2012.

[94] 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 - To Appear)*. IEEE, 2020.

[95] M. Imani, T. Nassar, A. Rahimi, and T. Rosing, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*. IEEE, 2018, pp. 271–274.

[96] 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, 2018.

[97] 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.

[98] P. Neubert, S. Schubert, and P. Protzel, "An introduction to hyperdimensional computing for robotics," *KI-Künstliche Intelligenz*, vol. 33, no. 4, pp. 319–330, 2019.

[99] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 52.

[100] 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)*. IEEE, 2017, pp. 1–8.

[101] 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)*. IEEE, 2019, pp. 190–198.

[102] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 53–62.

[103] 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*, 2019.

[104] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 4, pp. 1–25, 2019.

[105] 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)*. IEEE, 2019, pp. 126–131.

[106] "Uci machine learning repository," http://archive.ics.uci.edu/ml/datasets/ISOLET.

[107] "7 series fpgas data sheet," Data Sheet, Xilinx, February 2108.

[108] M. Imani, S. Salamat, S. Gupta, J. Huang, and T. Rosing, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 493–498.

[109] "Uci machine learning repository," https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones.

[110] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits, 1998," *URL http://yann. lecun. com/exdb/mnist*, vol. 10, p. 34, 1998.

[111] "Product brief igloo2 fpga," Product Brief, Microsemi, August 2018.

[112] 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 *55th Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[113] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini *et al.*, "In-memory hyperdimensional computing," *Nature Electronics*, pp. 1–11, 2020.

[114] M. Imani, Y. Kim *et al.*, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1591–1594.

[115] P. Alonso *et al.*, "Hyperembed: Tradeoffs between resources and performance in nlp tasks with hyperdimensional computing enabled embedding of n-gram statistics," in *International Joint Conference on Neural Networks*. IEEE, 2021.

[116] "Uci machine learning repository," https://archive.ics.uci.edu/ml/datasets/.

[117] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[118] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1946–1956.

[119] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[120] L. Yang and B. Murmann, "Sram voltage scaling for energy-efficient convolutional neural networks," in *International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2017, pp. 7–12.

[121] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 2017.

[122] A. Ultsch, "Clustering with som: U^* c," in *Proceedings of the workshop on self-organizing maps, 2005*, 2005.

[123] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-dnn: An open framework for mapping dnn models to cloud fpgas," in *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, 2019, pp. 73–82.

[124] K. Ergun, R. Ayoub, P. Mercati, and T. Rosing, "Dynamic reliability management of multi-gateway iot edge computing systems," *IEEE Internet of Things Journal*, 2022.

[125] F. Mireshghallah, M. Taram, P. Vepakomma, A. Singh, R. Raskar, and H. Esmaeilzadeh, "Privacy in deep learning: A survey," *arXiv preprint arXiv:2004.12254*, 2020.

[126] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han *et al.*, "Mcunet: Tiny deep learning on iot devices," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 711–11 722, 2020.

[127] S. Zhang, R. Wang, J. J. Zhang, A. Rahimi, and X. Jiao, "Assessing robustness of hyperdimensional computing against errors in associative memory," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2021, pp. 211–217.

[128] J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, and T. Simunic, "hydrea: Utilizing hyperdimensional computing for a more robust and efficient machine learning system," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 6, pp. 1–25, 2022.

[129] B. Khaleghi, J. Kang, H. Xu, J. Morris, and T. Rosing, "Generic: highly efficient learning engine on edge using hyperdimensional computing," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1117–1122.

[130] A. Hérnandez-Cano, R. Cammarota, and M. Imani, "Prid: Model inversion privacy attacks in hyperdimensional learning systems," in *ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 553–558.

[131] F. Yang and S. Ren, "Adversarial attacks on brain-inspired hyperdimensional computing-based classifiers," *arXiv preprint arXiv:2006.05594*, 2020.

[132] D. Ma, J. Guo, Y. Jiang, and X. Jiao, "Hdtest: Differential fuzz testing of brain-inspired hyperdimensional computing," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 391–396.

[133] R. Wang and X. Jiao, "Poisonhd: poison attack on brain-inspired hyperdimensional computing," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 298–303.

[134] P. Courrieu, "Fast computation of moore-penrose inverse matrices," *arXiv preprint arXiv:0804.4809*, 2008.

[135] Y. Shen, S. Dasgupta, and S. Navlakha, "Algorithmic insights on continual learning from fruit flies," *arXiv preprint arXiv:2107.07617*, 2021.

[136] S. Salamat, A. Haj Aboutalebi, B. Khaleghi, J. H. Lee, Y. S. Ki, and T. Rosing, "Nascent: Near-storage acceleration of database sort on smartssd," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 262–272.

[137] A. Dutta, S. Gupta, B. Khaleghi, R. Chandrasekaran, W. Xu, and T. Rosing, "Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, 2022, pp. 281–286.

[138] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, 2010, pp. 51–60.

[139] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, "What can we learn privately?" *SIAM Journal on Computing*, vol. 40, no. 3, pp. 793–826, 2011.

[140] J. K. Weihong Xu and T. Rosing, "Accelerating few-shot learning on reram using hyperdimensional computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023.

[141] N. Papernot, S. Chien, S. Song, A. Thakurta, and U. Erlingsson, "Making the shoe fit: Architectures, initializations, and tuning for learning with privacy," 2019.

[142] (2020) CPU energy meter. [Online]. Available: https://github.com/sosy-lab/cpu-energy-meter/

[143] D. Anguita, A. Ghio, L. Oneto, X. Parra Perez, and J. L. Reyes Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proceedings of the 21th international European symposium on artificial neural networks, computational intelligence and machine learning*, 2013, pp. 437–442.

[144] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *2012 16th international symposium on wearable computers*. IEEE, 2012, pp. 108–109.

[145] "The cifar dataset," https://www.cs.toronto.edu/~kriz/cifar.html.

[146] M. Abo-Zahhad, M. Farrag, A. Ali, and O. Amin, "An energy consumption model for wireless sensor networks," in *5th International Conference on Energy Aware Computing Systems & Applications*. IEEE, 2015, pp. 1–4.

[147] 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 *et al.*, "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.

[148] C. Chen and J. Lee, "Stochastic adaptive line search for differentially private optimization," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 1011–1020.