



# Architecting High Performance Silicon Systems for Accurate and Efficient On-Chip Deep Learning

## Citation

Tambe, Thierry. 2023. Architecting High Performance Silicon Systems for Accurate and Efficient On-Chip Deep Learning. Doctoral dissertation, Harvard University Graduate School of Arts and Sciences.

## Permanent link

<https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37375806>

## Terms of Use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA>

## Share Your Story

The Harvard community has made this article openly available. Please share how this access benefits you. [Submit a story](#).

[Accessibility](#)

HARVARD UNIVERSITY  
Graduate School of Arts and Sciences




DISSERTATION ACCEPTANCE CERTIFICATE


The undersigned, appointed by the


Harvard John A. Paulson School of Engineering and Applied Sciences  
have examined a dissertation entitled:

“Architecting High Performance Silicon Systems for Accurate and Efficient On-Chip  
Deep Learning”

presented by: Thierry Tambe

Signature   
*Typed name:* Professor G. Wei

Signature   
*Typed name:* Professor D. Brooks

Signature   
*Typed name:* Professor S. Rush

Signature   
*Typed name:* Dr. B. Khailany

April 21, 2023

# Architecting High Performance Silicon Systems for Accurate and Efficient On-Chip Deep Learning

A DISSERTATION PRESENTED

BY

THIERRY TAMBE

TO

THE SCHOOL OF ENGINEERING AND APPLIED SCIENCES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN THE SUBJECT OF

ELECTRICAL ENGINEERING

HARVARD UNIVERSITY

CAMBRIDGE, MASSACHUSETTS

APRIL 2023

©2023 – THIERRY TAMBE  
ALL RIGHTS RESERVED.

# Architecting High Performance Silicon Systems for Accurate and Efficient On-Chip Deep Learning

## ABSTRACT

The unabated pursuit of omniscient and omnipotent AI is levying hefty latency, memory, and energy taxes at all computing scales. At the same time, the twilight of Dennard scaling means traditional performance gains are no longer proportionally attained with reduction in transistor feature size – compelling a global trend towards application-based hardware specialization.

Over the course of my PhD, I have built a heterogeneity of solutions co-optimized across the algorithm, architecture, and silicon stack to generate breakthrough advances in arithmetic performance, compute density and flexibility, and energy efficiency for on-chip machine learning (ML), and natural language processing (NLP) in particular. My work aims to significantly increase the application space of embedded ML computing, in both the inference and training regimes, by coalescing innovative vectors spanning the algorithm, memory subsystem, hardware architecture, and circuit layers, while tuning their designs and inter-dependencies to promote greater performance, energy efficiency, and reliability within a silicon chip system.

In the algorithm front, this thesis discusses best paper award-winning work on a novel floating-point based data type, *AdaptivFloat*, which enables resilient quantized AI computations; and is particularly suitable for NLP networks with large parameter distribution. To evaluate *AdaptivFloat* impact on a real system, this thesis describes a 16nm chip prototype that integrates *FlexASR*, a programmable hardware accelerator with *AdaptivFloat*-based processing elements, and specialized for attention-based recurrent neural networks used in speech and machine translation AI workloads.

We further verify FlexASR fidelity to the front-end AI application via a formal hardware/software compiler interface.

Towards the goal of lowering the prohibitive energy cost of inferencing large language models on TinyML devices, this dissertation describes a principled algorithm-hardware co-design solution, validated in a 12nm chip tapeout, that accelerates Transformer workloads by tailoring the accelerator’s latency and energy expenditures according to the complexity of the input query it processes.

Finally, recognizing that the overwhelming majority of the data generated during the deep learning training process exhibits a very short-lived lifetime, this thesis proposes leveraging non-conventional embedded dynamic RAMs (eDRAMs) as the main on-chip storage medium for ML training data – which, along with a tightly-coupled offering of algorithmic alterations and custom hardware specialization, yields significant energy efficiency advantages over conventional SRAMs.

# Contents

TITLE PAGE	i
COPYRIGHT	ii
ABSTRACT	iii
TABLE OF CONTENTS	vi
PREVIOUS WORK	vii
DEDICATION	ix
ACKNOWLEDGMENTS	x
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 The Grand Challenge to Meet Tomorrow’s Needs for Information Processing . . .	2
1.2 The Need for Application-to-Silicon Cross-Stack Co-Design . . . . .	4
1.3 Thesis Contributions . . . . .	6
1.4 Thesis Roadmap . . . . .	10
<b>2 BACKGROUND AND RELATED WORK</b>	<b>12</b>
2.1 Foundational Speech and NLP Neural Networks . . . . .	13
2.2 Prominent Number Systems for Deep Learning . . . . .	19
2.3 Related Work . . . . .	21
<b>3 ENABLING ADAPTIVE AND ENERGY-EFFICIENT QUANTIZED COMPUTATIONS</b>	<b>25</b>
3.1 AdaptivFloat: an Adaptive Floating-Point Based Number System for Accurate and Resilient DNN Inference . . . . .	27
3.2 Experimental Results . . . . .	29
3.3 Hardware Implementation . . . . .	34
3.4 Results and Implications . . . . .	35

3.5	Facilitating Data Type Exploration for Efficient Hardware Design . . . . .	36
3.6	Takeaways . . . . .	38
3.7	Follow-on Research . . . . .	38
<b>4</b>	<b>ACCELERATING EDGE AI ATTENTION-BASED SPEECH-TO-TEXT RNNs</b>	<b>40</b>
4.1	The SM6 SoC Architecture . . . . .	43
4.2	The FlexASR Hardware Accelerator . . . . .	48
4.3	Post-Silicon Measurement Results . . . . .	53
4.4	Verifying FlexASR via a Formal Hardware/Software Compiler Interface . . . . .	59
4.5	Takeaways . . . . .	62
<b>5</b>	<b>LOWERING THE COST OF INFERRING LARGE LANGUAGE MODELS ON EMBEDDED DEVICES</b>	<b>64</b>
5.1	Entropy-based Early Exit . . . . .	66
5.2	Entropy-Controlled Voltage-Frequency Scaling . . . . .	69
5.3	The EdgeBERT Sparse Transformer Processor . . . . .	70
5.4	12nm Chip Prototype and Post-Silicon Measurement Results . . . . .	74
5.5	Takeaways and Follow-on Research . . . . .	77
<b>6</b>	<b>CAMEL: CO-DESIGNING AI MODELS AND eDRAMs FOR EFFICIENT ON-CHIP LEARNING</b>	<b>78</b>
6.1	Using eDRAMs as the Main Storage Medium for On-Device Training . . . . .	80
6.2	Computations in DNN Training . . . . .	82
6.3	Reversible DNN Architectures . . . . .	83
6.4	Proposing a 2D Block Floating-Point Datatype . . . . .	87
6.5	The CAMEL Hardware Accelerator System with a Hybrid eDRAM-SRAM Memory Subsystem . . . . .	88
6.6	Evaluation . . . . .	93
6.7	Conclusion and Future Work . . . . .	101
<b>7</b>	<b>CONCLUSION</b>	<b>102</b>
7.1	Future Research Directions . . . . .	103
	<b>REFERENCES</b>	<b>122</b>



# Previous work

Portions of this dissertation appear in the following works:

**Thierry Tambe**, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, David Brooks, Gu-Yeon Wei. “Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference”. ACM/IEEE Design Automation Conference (DAC 2020).

**Thierry Tambe**, En-Yu Yang, Glenn Ko, Yuji Chai, Coleman Hooper, Marco Donato, Paul Whatmough, Alexander Rush, David Brooks, Gu-Yeon Wei. “A 25mm<sup>2</sup> SoC for IoT Devices with 18ms Noise-Robust Speech-To-Text Latency via Bayesian Speech Denoising and Attention-based Sequence-to-Sequence DNN Speech Recognition in 16nm Finfet”. IEEE International Solid-State Circuits Conference (ISSCC 2021).

**Thierry Tambe**, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul Whatmough, Alexander Rush, David Brooks, Gu-Yeon Wei. “EdgeBERT: Sentence-Level Energy Optimizations for Latency-Aware Multi-Task NLP Inference”. IEEE/ACM International Symposium on Microarchitecture (MICRO 2021).

**Thierry Tambe**, En-Yu Yang, Glenn Ko, Yuji Chai, Coleman Hooper, Marco Donato, Paul Whatmough, Alexander Rush, David Brooks, Gu-Yeon Wei. “A 16-nm SoC for Noise-Robust Speech and NLP Edge AI Inference with Bayesian Sound Source Separation and Attention-based DNNs”. IEEE Journal of Solid-State Circuits (JSSC 2022).

Abdulrahman Mahmoud, **Thierry Tambe**, Tarek Aloui, David Brooks, Gu-Yeon Wei. “GoldenEye: A Platform for Evaluating Emerging Numerical Data Formats in DNN Accelerators”. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2022).

Cheng Tan, **Thierry Tambe**, Jeff Zhang, Bo Fang, Tong Geng, Gu-Yeon Wei, David Brooks, Antonino Tumeo, Ganesh Gopalakrishnan, Ang Li. “ASAP: Automatic Synthesis of Area-Efficient and Precision-Aware CGRAs”. ACM International Conference on Supercomputing (ICS 2022).

Bo-Yuan Huang, Steven Lyubomirsky, Yi Li, Mike He, **Thierry Tambe**, Gus Henry Smith, Akash Gaonkar, Vishal Canumalla, Gu-Yeon Wei, Aarti Gupta, Zachary Tatlock, Sharad Malik. “Specialized Accelerators and Compiler Flows: Replacing Accelerator APIs with a Formal Software/Hardware Interface”. arXiv preprint arXiv:2203.00218 (2022).

**Thierry Tambe**, Jeff Zhang, Coleman Hooper, Tianyu Jia, Paul Whatmough, Jeff Zuckerman, Maico Cassel, Erik Loscalzo, Davide Giri, Kenneth Shepard, Luca Carloni, Alexander Rush, David Brooks, and Gu-Yeon Wei, “A 12nm 18.1 TFLOPs/W Sparse Transformer Processor with Entropy-based Early Exit, Mixed-Precision Predication and Fine-Grained Power Management”. IEEE International Solid-State Circuits Conference (ISSCC 2023).

Sai Zhang\*, **Thierry Tambe\***, Nestor Cuevas, Jeff Zhang, Gu-Yeon Wei, David Brooks. “CAMEL: Co-designing AI Models and Embedded DRAMs for Efficient On-Device Learning”. arXiv preprint arXiv:2305.03148 (2023). \**Authors contributed equally.*

TO MY DEAR PARENTS: CLEMENTINE TAMBE AND JUSTIN TAMBE.

# Acknowledgments

This dissertation was only possible with the concerted effort of a lot of people supporting me over the last five and half years. I am deeply grateful to all of them for their encouragement, advice, and contributions throughout my PhD journey.

I would like to start by thanking my advisors and mentors. First, **Gu-Yeon Wei** and **David Brooks**. Despite having more of an engineering bent after a relatively long spell in the industry, you took a chance on me and sharpened me as a researcher, writer, and communicator. Notably, you put the right pieces in place to allow me to do my best possible work. Thank you both for your prescient guidance and mentorship. **Sasha Rush**, thank you for being a wonderful machine learning mentor. I am very appreciative of the numerous conversations we had, discussing and elucidating NLP stuff, as they informed a lot of key decisions during the design of our AI hardware processors. **Brucek Khailany**, thank you for being an amazing collaborator on the DARPA CRAFT program and for allowing me to intern in your group at NVIDIA. Working with you, Rangha Venkatesan, Sophia Shao, and Ben Keller, in the early stages in my PhD, catalyzed a lot of subsequent research explorations, besides making me an avid HLS enthusiast. **Paul Whatmough**, **Marco Donato**, **Glenn Ko**, and **Sae Kyu Lee**, I feel quite lucky for having worked closely with you throughout my PhD. Your mentorship on so many aspects of chip development, as well as the lively discussions we had entertaining the viability of different research and engineering ideas – have had a profound effect on my research imprint. Many features of my work have built on the seeds you have sown. So, I owe it to the four of you for providing a launchpad that allowed me to be productive during my time at Harvard. **Vijay Reddi**, **Demba Ba**, and **Gage Hills**, thank you for fielding me very thoughtful and practical feedback during my PhD journey, which really helped me internalize and approach my work with greater confidence, and resist instincts to sell myself short.

**Glenn Holloway**, it is quite difficult to gauge the extent of the impact you have had on my PhD career. Whether it is fixing machine failures and tool issues, setting up many of the logistics surrounding the chip development and testing infrastructure, answering my calls regardless of the time in the day or night, you went above and beyond to make sure I had everything I needed to do my best work – not to mention that I learned a great deal from witnessing your IT wizardry. Your tireless dedication to each of us in the lab has been remarkable. You are truly a hero to me.

I am thankful for having worked and hung out with a brilliant group of colleagues and collaborators at Harvard. **En-Yu (Daniel) Yang**, your technical brilliance and the elegance through which you approached problem-solving were amazing to behold. I could not have asked for a better col-

laborator on the inaugural FlexASR project. **Coleman Hooper**, your highly-motivated drive and noble work ethics created breakthroughs on the EdgeBERT project. Thank you for being such a strong collaborator. **Lillian Pentecost**, working with you and bringing the NVM angle to the EdgeBERT project has been enriching. Your methodical approach to dissecting problems and motivating proposed solutions has been inspiring. **Udit Gupta**, I owe my first immersion into the science of cutting-edge research to you as being part of the MASR project gave me a plethora of learnings that I carried forward in my subsequent PhD projects. Thank you for being an amazing and generous graduate student mentor. I also cherished the light moments and coffee break outings you, Lillie, and I shared together when we were still stationed at Maxwell Dworkin. **Siming Ma**, thank you for providing very helpful and actionable advice on several of my chip tapeouts. Your feedback saved a lot of engineering efforts and helped us meet aggressive shuttle deadlines. **Tianyu Jia**, it has been such a treat and privilege to collaborate with you on EdgeBERT and Epochs-1. Watching you tackle big research and engineering problems with poise and humility has been quite inspiring. **Jeff Zhang**, I am thankful for your eagerness to contribute time and efforts to some of our most challenging problems. Thank you for contributing to the CAMEL path-finding efforts and for pioneering the compiler frontend to EdgeBERT. **Sai Qian Zhang**, it has been a privilege and pleasure to collaborate closely with you on the CAMEL project. It has been inspiring to see your research acumen elevate the refinement of this work in so many remarkable ways. **Nestor Cuevas**, thank you for the strong work ethics and the highly-motivated drive you brought to the CAMEL project. I have learned a great deal from watching you translate eDRAM research proposals into real and robust implementations. **Abdulrahman Mahmoud**, working with you on GoldenEye has been truly gratifying. Thank you also for being for a supportive ear whenever I needed it and sharing break times with me over games of ping-pong.

Thank you to my amazing collaborators at Columbia University (**Joseph Zuckerman, Maico Cassel, Erik Loscalzo, Davide Giri, Kenneth Shepard, Luca Carloni**) and IBM (**Martin Cochet, Karthik Swaminathan, Pradip Bose**) with whom I shared an incredibly enriching and rewarding experience bringing the DARPA Epochs-1 chip from an intimidating concept to a present reality.

Thank you to my brilliant collaborators at the SRC JUMP ADA center from Princeton University (**Bo-Yuan Huang, Yi Li, Akash Gaonkar, Aarti Gupta, Sharad Malik**) and the University of Washington (**Steven Lyubomirsky, Mike He, Gus Smith, Vishal Canumalla, Zachary Tatlock**) with whom I shared a very productive and collaborative spirit in the daunting quest to close the mapping gap between domain-specific languages and custom hardware accelerators.

Thank you **Sandeep Garg** for always being on standby and eager to debug and resolve, promptly, any Catapult/HLS issues we encountered. Your leadership helped us achieve high quality silicon in our various chip tapeouts.

I would also like to thank the Harvard Graduate Christian Community student group, which I have been a member of during the course of my PhD, for providing a sense of community and social support, especially during the zenith of the covid-19 pandemic.

Finally, I thank Harvard SEAS, the NVIDIA Graduate Fellowship Program, the SRC JUMP ADA center, and DARPA for funding the research covered in this dissertation. The views, findings, and/or conclusions expressed in this dissertation are those of the author and should not be inter-

preted as representing the official views of any funding agency.

# 1

## Introduction

The semiconductor industry has benefited from years of productive returns by investing in the promises of Moore's Law. Coupled with frequency and voltage scaling, consistent and tangible performance improvements were reaped across several generations of computing systems. However, as transistor miniaturization reached nanoscale dimensions, the combined effects of the larger than expected oxide leakage current and junction overheating, and the fact that interconnects could not proportionally track with the downsizing rate of FinFETs – ultimately increased the on-chip

power density and made it difficult to continue scaling down transistors without hitting the proverbial “Power Wall”. Dennard’s Law effectively ended. Faced with this acute challenge, the chip industry transitioned to multicore scaling in the desperate quest to keep delivering energy efficiency gains. In theory, the idea of multi-core designs made sense. Silicon chips could continue packing more and more transistors by increasing the number of parallelized cores, that would operate at a lower clock frequency than a single core (to avoid thermal failures), but would generate higher aggregate throughput per unit area. However, in practice, this solution led computer chips to hit a “Utilization Wall” as many cores were forced to be either completely turned off or significantly underclocked to provide an edge in power efficiency<sup>137</sup>. Therefore, while multicore processors effected considerable progress in SoCs’ peak performance, they have not been an end-all solution to power density challenges. The problem of dark silicon, along with Amdahl’s Law constraints, communication overheads among cores, core synchronization overheads, software and programming challenges, have all contributed to the continued search for new strategies to improve computing performance in the post-Dennard Scaling era.

It is imperative to address the wide-ranging implications stemming from the breakdown of Dennardian scaling considering that we may not be adequately prepared for the looming deluge of data spurred by the surging democratization of AI and other emerging compute-intensive applications.

## 1.1 THE GRAND CHALLENGE TO MEET TOMORROW’S NEEDS FOR INFORMATION PROCESSING

Semiconductor technologies are facing a reckoning. The coming decades will see an exponential increase in the amount of data that will need to be moved, stored, computed, and communicated to the end user. Trends and projections from the world sensor data are illustrated in Figure 1.1<sup>119</sup>, showing that by 2032, data production per sensor will reach an estimated  $10^{27}$  bytes, far surpassing



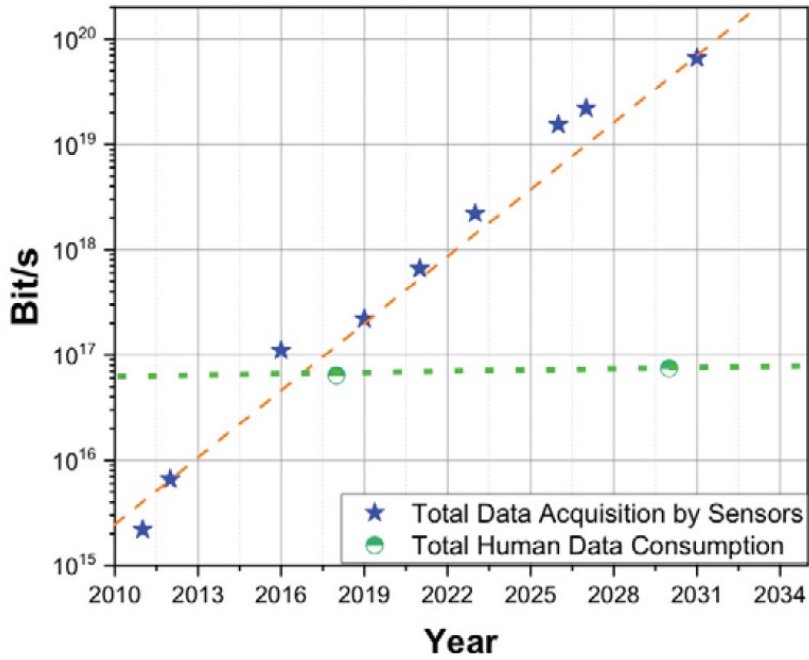


Figure 1.1: The explosion of sensor data within the next decade will far exceed humans' capacity to process and make effective use of it. Source: Semiconductor Research Corporation 2020 Decadal Plan<sup>119</sup>.

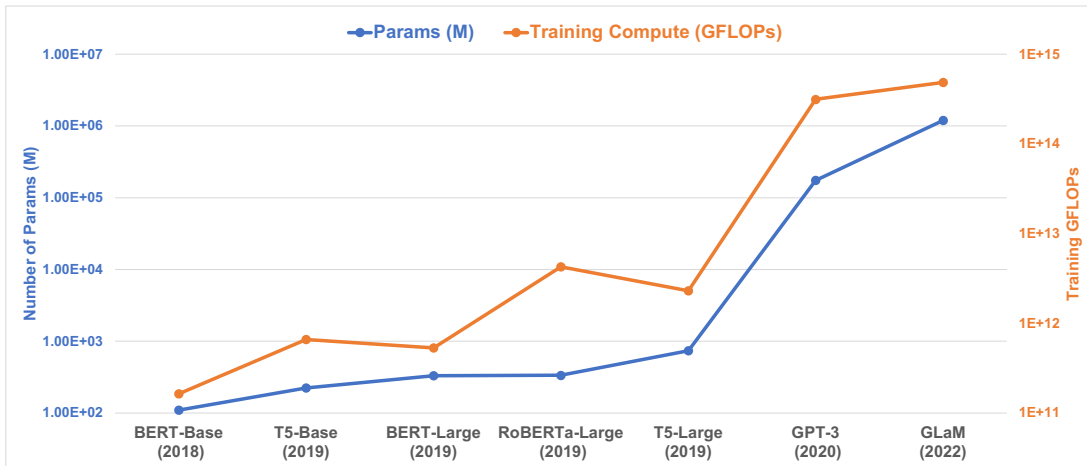


Figure 1.2: Large language models have been scaling at an exponential rate in terms of memory footprint and training compute over the last five years<sup>10</sup>.

humans' ability to effectively digest and process it. This raises an urgent call to dramatically change the current computing paradigm as Tomorrow's needs for information processing, propelled by emerging data-intensive applications such as deep learning, AR/VR, autonomous driving – are simply unachievable with the current state of semiconductor technologies. Even more concerning, the unrelenting pursuit for greater linguistic understanding and representation is currently sparking a worldwide AI arms race, driving natural language processing (NLP) models to extreme scales. The amount of AI computations required to produce advanced generative models is doubling every six months<sup>121</sup>, outpacing the steepest slope of Moore's Law. Putting this scale into perspective, recent large language models are now exceeding 1 trillion parameters (e.g., GLaM<sup>31</sup>, Switch Transformers<sup>33</sup>), and requiring trillions of training FLOPs as shown in Figure 1.2.

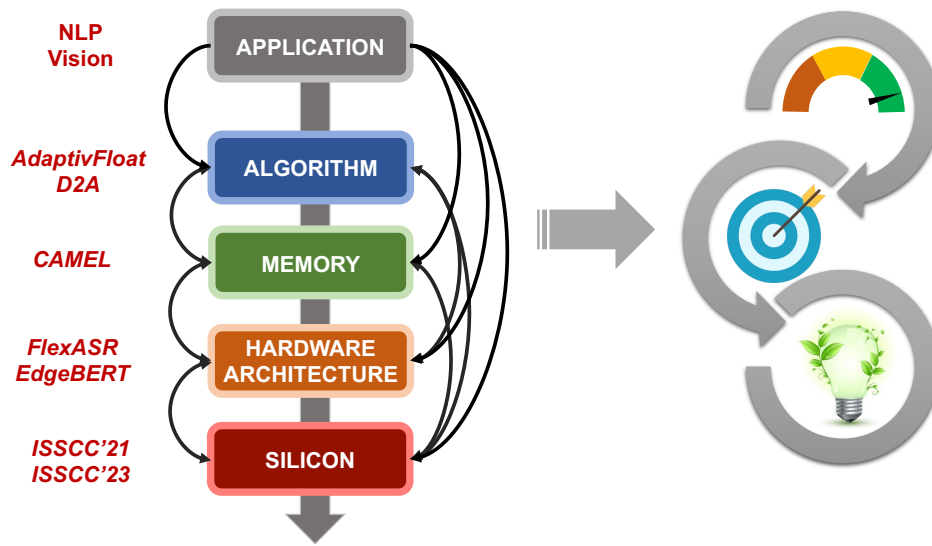
As I see it, the most effective approach to optimize the growing cost of computing, in a sustainable manner, is to exploit the very same data currently driving the ongoing AI revolution. Solid-state based solutions, alone, such as further CMOS shrinking, in-memory computing, or advanced packaging techniques, in my view, can only provide temporary relief. However, the combination of these sophisticated circuit and packaging advancements along with application-driven or data-driven measures could deliver a greater value proposition in terms of compute and energy efficiency.

## 1.2 THE NEED FOR APPLICATION-TO-SILICON CROSS-STACK CO-DESIGN

The coming AI wave is motivating a gradual shift away from compute-centric processing and towards data-centric processing by leveraging intrinsics in the underlying application in order to boost performance and energy efficiency at all computing scales. Concurrently, privacy and security concerns, as well as, stricter latency requirements are incentivizing a growing deployment of AI workloads to edge devices, targeting both inference and training.

Processing at the edge is challenging for several reasons.

1. The on-chip memory capacity is usually acutely insufficient to satisfy the storage needs of the AI application. This then compels off-chip data movements to external memories, which unfortunately are characterized by longer round-trip latencies and significantly higher energy consumption (e.g., energy cost of accessing DRAM is more than  $100\times$  larger compared to on-chip accesses<sup>53</sup>).
2. Relatedly, the limited amount of on-chip computing resources applies a restrictive cap on the application space that can be deployed to cloudless edge devices. Tangibly, smaller AI models with lower accuracy are often run on these low-capacity edge platforms, and the on-device execution is typically focused on the inference scenario exclusively.
3. As CMOS scaling does not provide proportional gains in I/O and interconnect bandwidth, on-chip data movements typically account for the majority of energy expenditures. An on-chip memory hierarchy exploiting spatial and temporal locality can improve the SoC energy efficiency by reducing latencies and increasing data access efficiency. However, we also need to pay attention to accelerator-CPU data exchanges, which may become excessive in a naive ASIC implementation. Several processing-in-memory or near-memory computing techniques have been proposed<sup>36</sup> to more aggressively address these “Memory Wall” challenges.
4. In order to acquire advantages in latency and energy, the computation is executed in very specialized hardware, which complicates the development of the compiler ecosystem for lowering the application code down to the edge device. Many application-specific hardwares are designed with customized, non-traditional instruction set architectures (ISAs), requiring dedicated software drivers that only the hardware designer understands. This, as a result, inhibits community adoption of specialized AI hardware accelerators.
5. The field of deep learning is exhibiting very fast growth and maturity with newer AI mod-



**Figure 1.3:** My PhD research has developed novel algorithms, memory systems, specialized architectures, and silicon chips, while tuning their designs and inter-dependencies to promote greater performance, efficiency, and reliability.

els being released at an increasingly rapid pace. Therefore, designers must ensure that their hardware does not become obsolete as the AI algorithm evolves. The challenge is to provide an optimized balance between hardware reconfigurability and flexibility without sacrificing performance and energy efficiency.

In light of the above-mentioned challenges, my PhD research has sought to drastically increase the application space of embedded AI computing by coalescing a hierarchical heterogeneity of innovative vectors across the computation stack, from algorithms down to the silicon, in order to accelerate the future of computing in the post-CMOS era. To that end, the next section outlines the research contributions made during my PhD journey in order to address some of these issues.

### 1.3 THESIS CONTRIBUTIONS

Over the course of my PhD, I have built cross-stack solutions and further tuned their inter-dependencies in order to generate breakthrough advances in arithmetic performance, compute density, energy ef-

efficiency, as well as reliability (Figure 1.3). This cross-layer optimization, notably, involves several co-designs to address specific challenges in edge AI computing. Specifically:

1. **AdaptivFloat** which is a **co-design of the machine learning model and the number system**. Recognizing that parameters from NLP models exhibit a much wider dynamic range compared to those of CNN models, AdaptivFloat offers a floating-point based mathematical formulation that is adaptive to the statistical distribution of DNN parameters while being resilient to aggressive bitwidth compression.
2. **FlexASR** which is a **co-design of the machine learning model and the SoC architecture** in order to reap, on-chip, the benefits of highly accurate and noise-resilient speech-to-text AI models without incurring undue energy overheads. Prominently featured in a 16nm 2.5mm<sup>2</sup> SoC is the FlexASR hardware accelerator for efficient computations of attention-based RNNs, as well as, a Bayesian engine for on-chip real-time denoising.
3. **EdgeBERT** which, at the core, is a **co-design of the machine learning model and the NLP computation** in order to drastically reduce the latency and energy overheads of Transformer-based large language models via early exit, and per-sentence voltage frequency scaling. This work is further validated in a 12nm chip prototype.
4. **CAMEL** which is fundamentally a **co-design of the machine learning model and the memory subsystem** for efficient on-device training. Identifying on-chip memory accesses as a major energy efficiency bottleneck, we leverage embedded DRAMs (eDRAMs) instead of SRAMs as the main on-chip storage medium for the majority of DNN training parameters, which characteristically exhibit a transient profile.

Furthermore, I have had the great privilege to rigorously evaluate the above-mentioned co-designs in 12nm and 16nm chip tapeouts in order to reveal the true value of the full-stack optimization. As

we are moving towards an era of personalized computing, these different co-designs offer several learnings regarding the research rationale, and how one may approach a full system implementation with tangible performance and efficiency benefits. I summarize, at a very high-level, some key takeaways below:

- Emerging AI models exhibit a parameter distribution spread that far exceeds the dynamic range of commonly used fixed-point data types. Therefore, it has become imperative to provide a low precision numerical encoding solution that adaptively scales its representation range to cater to the most important or most impactful regions in the deep learning tensor (e.g., weights, activations, gradients). In `AdaptiveFloat`, this is accomplished by introducing a variable bias in the exponent field of the floating-point datatype.
- The proliferation of conversational AI interfaces is driving an exponential growth in the consumer IoT and wearable market. As they are featured in devices with small form factors (e.g., AR/VR glasses), it is of paramount importance that they work seamlessly in polyphonic or acoustically challenged environments. Rather than integrating a front-end noise cancelling circuit or scaling up the size of the speech-to-text DNN model to achieve noise robustness, which may be unoptimal considering the strict power budget of the IoT device, we demonstrate a more energy-efficient compromise by executing on-chip Bayesian based denoising prior to the attention-based automatic speech recognition (ASR) operation. The `FlexASR ML-SoC` co-design is an example of how next-generation system-on-chips may be architected leveraging recent advances in machine learning to obviate traditionally-adopted engineering solutions that may now be unsuitable in emerging technical specifications.
- The idea of dynamic voltage frequency scaling (DVFS) was initially proposed in the early 1990s and it took many years until we saw its deployment on Intel CPUs in the early 2000s. Fast forward 20 years, although DVFS has been the cornerstone to meeting application en-

ergy requirements, the problem of energy efficiency still exists, and in fact, has been made worse by the unrelenting pursuit of omniscient and omnipotent AI driven by large language models. In EdgeBERT, we identify the statistical metric of entropy as a springboard for finer-grained sentence-level power optimization. This ML-computation co-design, in my view, is a prelude to many future AI-aided smart power management schemes and circuits providing finer-grained control on energy and latency consumption in next-generation computing platforms.

- The shift towards data-centric processing will accelerate the preeminence of specialized computing, bringing forth more personalized experiences to everyday users. It will no longer be sufficient for edge devices to execute inference workloads. On-chip DNN training will soon become a first-order necessity to provide a more customized and intimate experience to users as they interact with the device. However, AI training on resource-limited ASICs is extremely difficult because of the intensive computing workload and the significant amount of on-chip memory consumption and traffic exacted by the neural network. As part of a ML-memory co-design, eDRAMs are utilized as the main storage carrier of DNN training parameters, providing at least twice the memory density compared to SRAMs. Furthermore, we leverage recent advances in reversible deep neural networks<sup>39</sup>, and co-design the DNN architecture along with the hardware system, and the scheduler to significantly shorten the data lifetime of parameters held in eDRAM – thereby avoiding periodic refresh. This co-design demonstrates more than  $2\times$  saving on total DNN training energy consumption compared to conventional baselines with external DRAMs, while achieving similar, and in some instances, better performance in validation accuracy.

The above-mentioned contributions are discussed with greater details in this dissertation. As a reference, the next section outlines the organization of the ensuing chapters.

## 1.4 THESIS ROADMAP

**(Chapter 2) Background and Related Work** introduces the foundational speech and natural language processing neural networks that have been the application focus for much of our algorithm-hardware co-designs. This chapter further reviews prior related work on algorithmic techniques and hardware architectures for efficient deep learning and NLP computations.

**(Chapter 3) Enabling Resilient and Energy-Efficient Quantized Computations** introduces a floating-point based mathematical blueprint, dubbed *AdaptivFloat*, which enables highly-accurate low-precision computations without compromising validation accuracy. Notably, in this work, we propose adaptive tensor scaling via exponent bias shift at a DNN layer granularity, which consistently produces higher inference accuracies compared to block floating-point, integer, IEEE-like float and posit encodings at low bit precision ( $\leq 8$ -bit) across a diverse set of state-of-the-art DNNs.

**(Chapter 4) FlexASR: Accelerating Edge AI Attention-Based Speech-to-Text RNNs** describes a 16nm SoC that executes a full speech-enhancing automatic speech recognition (ASR) pipeline using prominent advancements in machine learning. The proposed pipeline pre-processes incoming noise-corrupted speech using Arm A53 cores, then denoises the signal in a Markov Source Separation Engine (MSSE) accelerator, and finally accelerates attention-based sequence-to-sequence (seq2seq) ASR workloads in the FlexASR accelerator. Particular attention will be given to the architecture of the FlexASR processor which enables the system to achieve a peak efficiency of 7.8TFLOPs/W with real-time throughput.

**(Chapter 5) Lowering the Cost of Inferencing Large Language Models on Embedded Devices** describes a principled latency-driven approach to accelerate Transformer-based workloads on



edge devices with minimal energy consumption thanks to entropy-controlled early exit and voltage-frequency scaling. This work is further validated in a 12nm chip tapeout, demonstrating a  $6\times$  and  $7\times$  reduction in latency and energy, respectively, over the conventional inference using the popular NLP BERT model.

**(Chapter 6) Co-Designing AI Models and eDRAMs for Efficient On-Chip Learning** introduces our eDRAM-based fully-on-chip DNN training methodology. We enable the usage of eDRAM as an energy-efficient and viable on-chip memory for DNN training via three techniques that collectively lower the data lifetime of parameters held in eDRAM. At the hardware level, (1) a 2D Block Floating-Point (BFP) data type reduces the amount of computations per unit area, wherein the higher throughput translates to shorter data lifetime requirements and avoids retention failures. At the algorithm level, (2) we reformulate the computation graph of the DNN model to significantly shorten data lifetime and to relax storage requirements. (3) A scalable systolic array architecture with a hybrid eDRAM-SRAM memory subsystem is proposed to maximize the synergistic benefits of the algorithms designed to avoid periodic eDRAM refresh.

# 2

## Background and Related Work

THE FIELD OF DEEP LEARNING IS EVOLVING AT A DIZZYING PACE. There has been so much progress over the last ten years that a comprehensive survey of these advancements is beyond the scope of this dissertation. Instead, I will provide a background on the algorithms and solid-state solutions pertaining to the subfield of NLP which has been the main application focus of my research work. It is important to note that the insights drawn from the algorithmic and hardware approaches

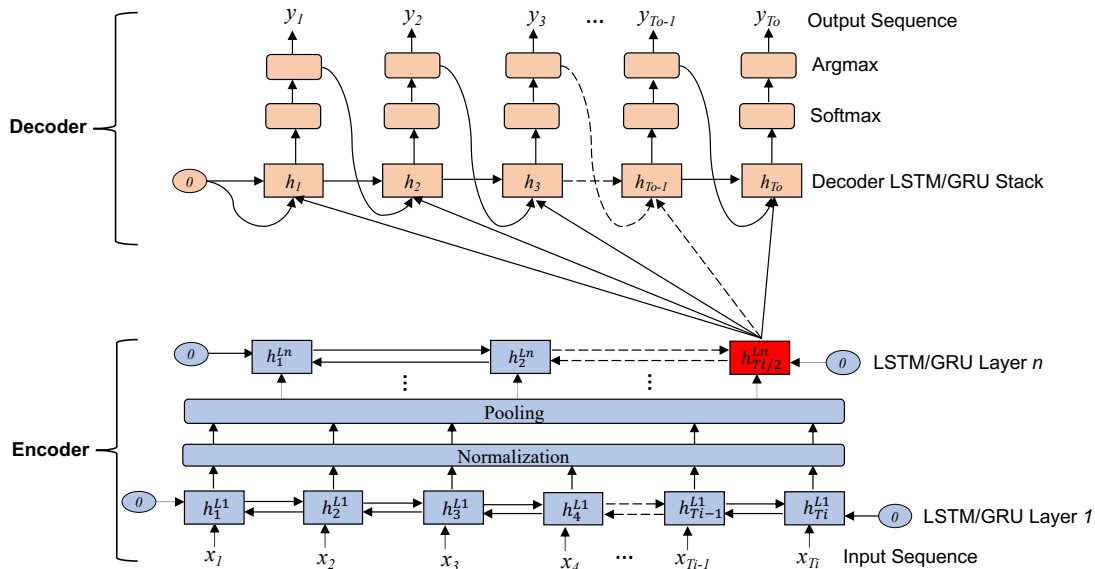
targeted towards NLP, are widely applicable to the rest of the deep learning field. Before recounting related work in Section 2.3, Section 2.1 first provides an overview of the neural networks which have catalyzed advancements in neural speech and language processing. Then, Section 2.2 introduces the main numerical data types used to encode deep learning parameters in specialized hardware.

## 2.1 FOUNDATIONAL SPEECH AND NLP NEURAL NETWORKS

Natural Language Processing (NLP) has a long history, dating back to the 1950s, during which rule-based methods were initially used to build systems for various tasks such as language translation, sentence analysis, and question answering. However, creating and managing rules quickly became labor-intensive, especially as the number of rules grew<sup>169</sup>. In the 1990s, the rapid growth of the internet brought about large amounts of open-source data that made it possible for statistical learning methods to work on such NLP tasks by identifying patterns and relationships between words and phrases. This led to a marked improvement in NLP performance. However, these statistical approaches struggled with complexities and subtleties of the human language such as nuances of meaning, and context learning.

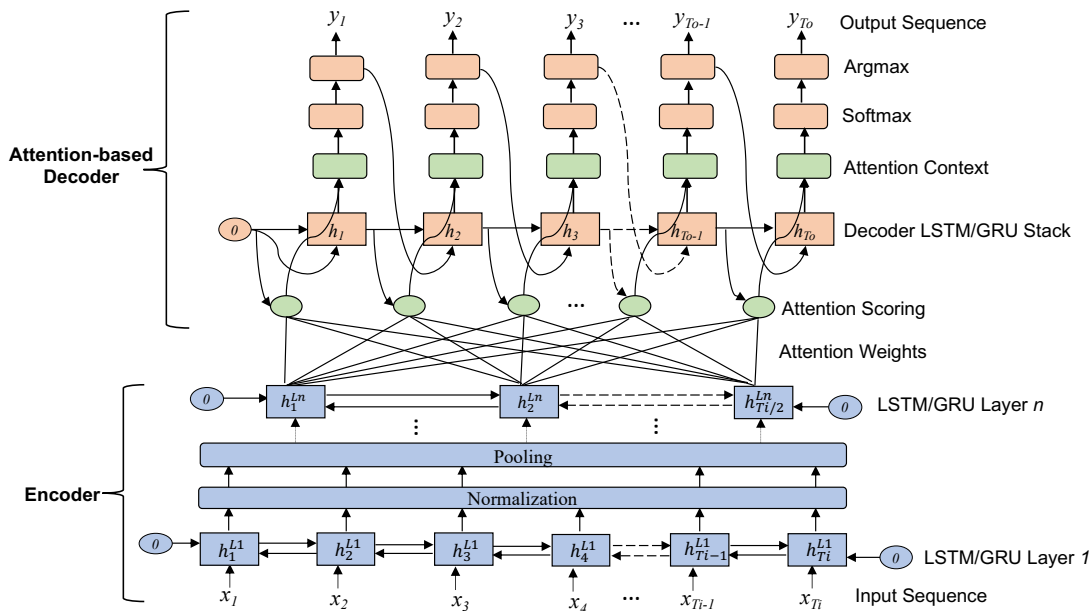
In the 2010s, deep learning approaches were introduced to speech and NLP tasks after demonstrating success in object recognition. Supervised learning using deep neural networks (DNNs), which are trained on large labeled datasets, rapidly outperformed statistical learning methods, achieving significantly better accuracy. This breakthrough in speech and NLP performance began with recurrent neural networks (RNNs) which were designed to model sequential data and, therefore, were well-suited for processing natural language. However, vanilla RNNs suffered from the problem of vanishing and exploding gradients, which made it challenging to train datasets containing long sequences.

To overcome this problem, researchers developed Gated Recurrent Unit (GRU) networks<sup>16</sup>, and



**Figure 2.1:** Original GRU/LSTM-based encoder-decoder architectures did not perform well in context learning because all the information from previous words was forced to be encoded inside a single context vector (highlighted in red in this figure) in the final layer of the encoder stack.

Long Short-Term Memory (LSTM) networks<sup>52</sup>. GRUs and LSTMs have become a popular choice for various neural speech and language representation problems, especially in speech-to-text (e.g., Deep Speech<sup>48,4</sup>), text-to-speech (e.g., Tacotron<sup>125</sup>, WaveNet<sup>146</sup>), and machine translation (e.g., Google NMT<sup>158</sup>, OpenNMT<sup>70</sup>) tasks. Given speech and NLP inference problems can be structured as sequence-to-sequence (seq2seq) processing, GRU and LSTM components were, at first, integrated in encoder-decoder architectures<sup>16</sup> as shown in Figure 2.1. The encoder stage contains unidirectional or bidirectional Vanilla RNN, GRU, or LSTM layers sandwiched between normalization and/or pooling layers. At each output timestep, the decoder stage estimates the relevance of each output timestep in an auto-regressive manner. However a major drawback of this topology is that the final hidden states (highlighted in red in Figure 2.1) were forced to contain the information of all the words in the source sentence, and then, should pass all this condensed information onto the decoder stage for auto-regressive prediction. Not surprisingly, this resulted in poorer per-



**Figure 2.2:** High-level architecture of an attention-based encoder-decoder neural network. For each output timestep, the decoder stage evaluates the relevance of all tokens in the encoder sequence via the attention mechanism.

formance when inferencing longer input sequences, and struggling even more in scenarios where different contexts were involved.

To address this problem, the attention mechanism<sup>8</sup> was proposed, and it is still today the most important ingredient of modern neural NLP.

### 2.1.1 THE ATTENTION MECHANISM

The attention mechanism<sup>8</sup> is a powerful technique that allows neural networks to emphasize the most relevant tokens of information in the source sequence when making predictions. This is especially important in situations where the input sequence is very long or complex, and the model needs to pay more attention to specific parts of the sequence that are most relevant for generating the output. The attention mechanism is perhaps the single greatest catalyst for the exceptional performance of modern NLP, and it has also been applied onto neural networks targeting computer

vision<sup>29,144</sup> tasks.

Figure 2.2 shows a typical attention-based seq2seq network known as a listen-attend-spell (LAS) model<sup>12</sup> \*. During each output time step, the decoder stage uses the attention mechanism to estimate the saliency weight of each output hidden state that emerges from the final encoder layer. This process is known as “soft” attention, as the final encoder hidden state sequence serves as a soft-addressable memory<sup>25</sup> whose words are weighted to compute the context vector after passing through the decoder LSTM/GRU stack. Assuming a greedy search, the most probable prediction is obtained by taking the Argmax of the Softmax output probabilities. We note that this Softmax operation is only beneficial for gradient approximation during training and can be effectively skipped during the inference computation. Also, the decoder of the LAS model predicts the next output time steps or tokens in an auto-regressive manner.

There are different types of attention mechanisms, but the most commonly used one is the dot-product attention mechanism, also known as the multiplicative attention mechanism, and which is notably adopted inside the popular Transformer architecture<sup>147</sup>. The dot-product attention is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.1)$$

where  $Q$  is the query (or encoder hidden state) vector being mapped to an output using a set of key-value pairs, represented by the vectors  $K$  and  $V$ , respectively.

The next section provides an overview of prominent NLP networks employing the dot attention mechanism via the Transformer model, which has become the most widely used neural building block for deep learning based NLP modeling.

---

\*Section 4 describes how these LAS models are efficiently computed in the FlexASR accelerator.

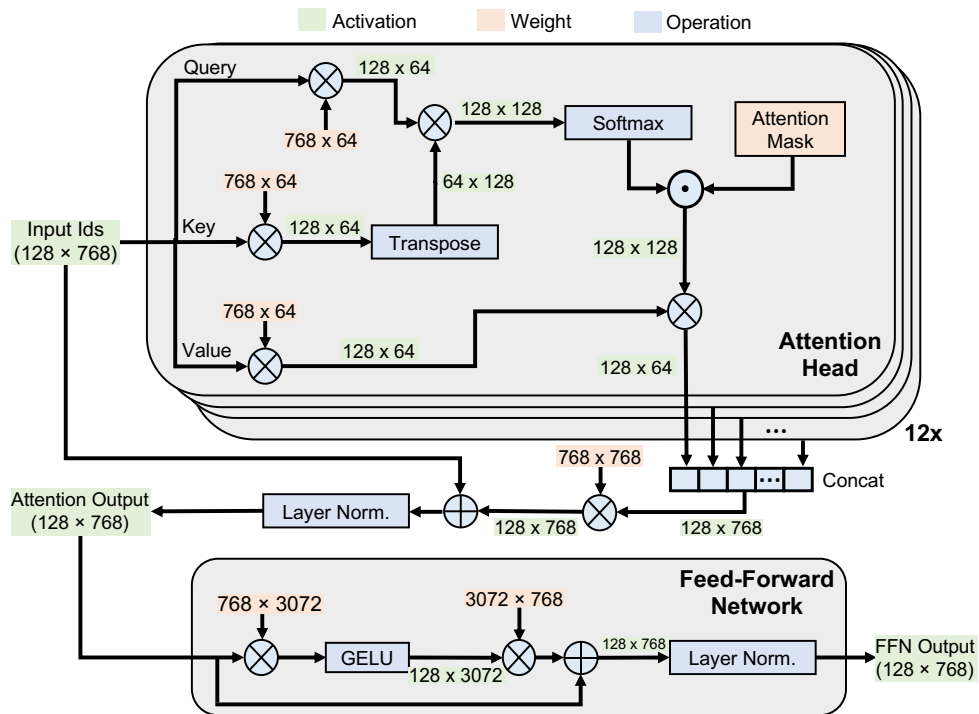


Figure 2.3: Computations inside a Transformer encoder. Here, the input sequence is composed of 128 tokens. To simplify the computational diagram, the bias layers are not included.

### 2.1.1.2 TRANSFORMER-BASED NLP NETWORKS

The Transformer architecture<sup>147</sup> has become the backbone of modern NLP and large language models<sup>†</sup>. Its computation diagram is shown Figure 2.3. A transformer typically comprises twelve parallel attention heads (often called the multi-head attention unit) whose outputs concatenate into an attention vector that feeds into a large feed-forward network. Transformers are designed to process entire sequences of words in parallel via the multi-head attention unit, wherein each attention head focuses on different parts of the input sequence. This enables more accurate predictions.

Perhaps, the most consequential language models have been BERT-based models<sup>27,84,78,116,130</sup>

<sup>†</sup>Section 5 describes how Transformer-based language models are efficiently computed in the EdgeBERT accelerator.

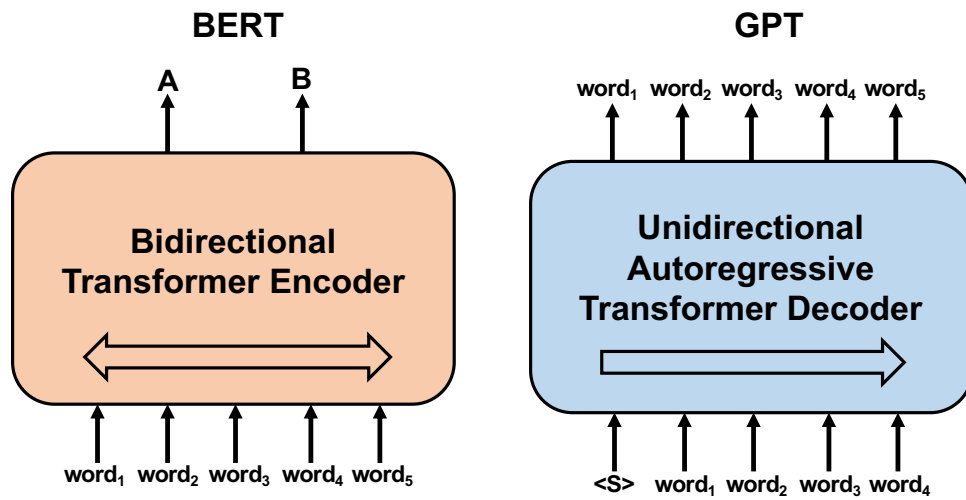
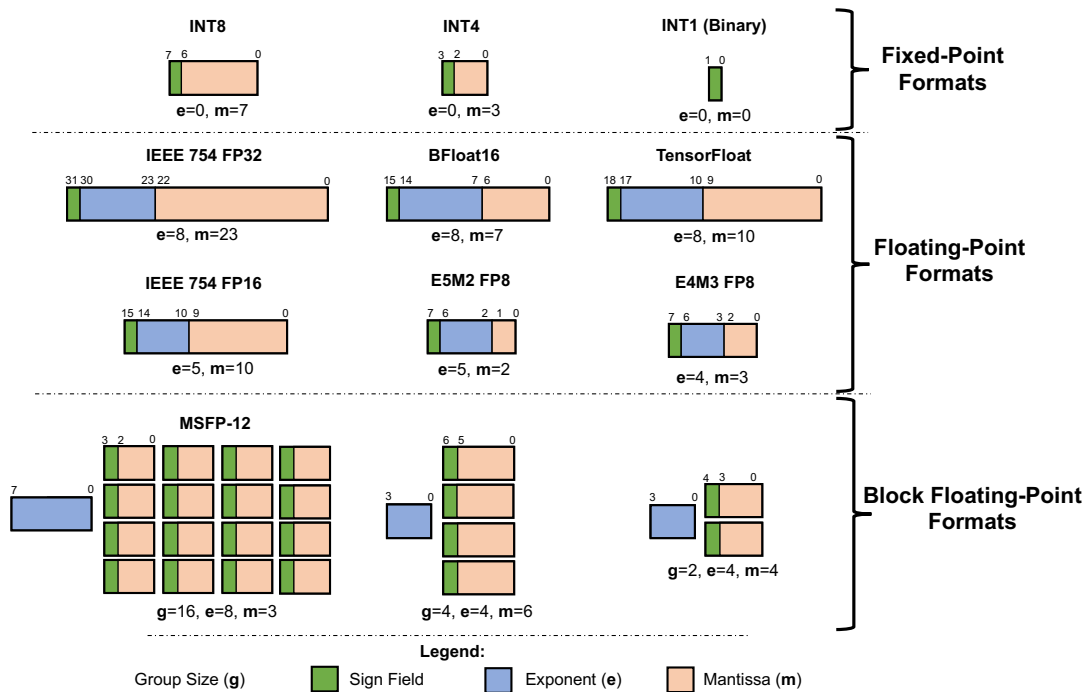


Figure 2.4: Illustration of architectural differences between BERT and GPT. BERT is trained to predict missing words from both directions while GPT is trained to generate words from left-to-right.

and the GPT family of models<sup>110,111,11</sup>. The main difference between BERT and GPT lies in their architecture (as shown in Figure 2.4) and fine-tuning objectives. It is a bidirectional model, meaning it is trained to predict missing words from both directions of a given text. BERT is pre-trained using a masked language modeling scheme whereby random words are masked and the model is trained to predict the masked words based on the surrounding words in the text. On the other hand, GPT is a unidirectional autoregressive model that is trained to generate text in a left-to-right manner. Another difference between BERT and GPT is in their fine-tuning objectives. BERT is commonly used for classification-based NLP tasks such as sentiment analysis, search, and text classification. It is fine-tuned using task-specific labeled data, where the objective is to minimize the classification error. In contrast, GPT is used for language generation tasks, such as text completion, summarization, and machine translation. It is fine-tuned by continuing to train the model on a specific generative task to produce high-quality and coherent output.

My research has uncovered that the performance of Transformer-based models is very sensitive to the choice of the number system and its underlying dynamic range. The next section provides an





**Figure 2.5:** Number formats commonly used during deep learning inference and training. Figure is adapted from FAST<sup>163</sup>.

overview of the numerical data types commonly used in deep learning and NLP in particular.

## 2.2 PROMINENT NUMBER SYSTEMS FOR DEEP LEARNING

Efficient computation in DNN training and inference largely depends on the number system used during matrix multiplications. The choice of the number system along with its underlying bit precision, in large part, dictate the arithmetic density and the energy efficiency of a computing platform. Figure 2.5 categorizes prominent data types used in deep learning into three groups: fixed point, floating point, and block floating-point (BFP). Each format is labeled with the number of exponent bits (e) and mantissa bits (m).

Fixed point formats do not have an exponent field, which simplifies the hardware but also limits

the dynamic range that can be represented. This format has been widely studied for DNN training and inference<sup>9,21,22,42,56,57,170</sup>. The smallest fixed point format is a 1-bit binary representation which is used in binarized neural networks, and does not have an exponent or mantissa bits.

On the other hand, floating-point formats offer wider dynamic range compared to fixed-point types. IEEE-754 32-bit floating point (or FP<sub>32</sub>) has traditionally been the prevalent number format due to its standardization and widespread usage in CPUs and GPUs. FP<sub>32</sub> has an 8-bit exponent field and a 23-bit mantissa field. A lower precision and standardized FP<sub>16</sub> version has a 5-bit exponent field and a 10-bit mantissa field. The growing democratization of AI has encouraged the research community to rethink the IEEE-754 floating-point number system, and has sought ways to alter this standard in order to better favor deep learning computational efficiency<sup>59</sup>. Therefore, we have seen custom floating-point formats emerged, such as Bfloat16<sup>41</sup> and TensorFloat<sup>97</sup>, which have been proposed by Google and NVIDIA, respectively. The semiconductor industry is also coalescing towards an efficient slate of FP8 formats known as HFP8<sup>129</sup>, which uses E<sub>4</sub>M<sub>3</sub> during inference (or forward pass computations), and E<sub>5</sub>M<sub>2</sub> during the backward pass computation as it has been found to be more robust during gradient computations.

Block Floating Point (BFP) provides an efficient compromise between fixed and floating point by using a single shared exponent to represent a tensor made of multiple scalars, which significantly reduces its memory footprint. BFP has received recent traction in deep learning and accelerator design due to its potential hardware performance optimization and the fact that it can encode numbers with a dynamic range as wide as standard floating-point. Prominent BFP formats include Flex-Point<sup>73</sup> which uses a 16-bit mantissa and a 5-bit shared exponent across an entire tensor. And, Microsoft notably proposed a BFP format, dubbed MSFP-12<sup>24</sup>, for DNN inference on their Project Brainwave cloud computing platform<sup>34</sup>. MSFP-12 uses a 3-bit mantissa field and a 8-bit shared exponent to represent a tensor of 16 scalars.

## 2.3 RELATED WORK

In this section, we cover notable work on the algorithm and hardware architecture fronts that promote high computational and energy efficiency during the AI and NLP on-device execution.

### 2.3.1 ALGORITHMS AND TECHNIQUES FOR EFFICIENT AI AND NLP COMPUTATIONS

The most common deep learning optimization techniques are pruning and quantization. They both aim to reduce the size and computational complexity of DNNs without or with modest accuracy degradation. Although these techniques were originally proposed to optimize convolutional neural networks (CNNs), they have been proven to be effective in Transformers as well. Chapter 5 further identifies opportunities to boost NLP energy efficiency via entropy-controlled early exit and latency-aware voltage-frequency scaling.

**Pruning** seeks to alleviate the computational complexity of DNNs by removing unimportant weights and neurons in the network, making it sparse. This sparsity can then be exploited during the hardware execution by avoiding or skipping the computation of the pruned parameters.

Various variations of pruning have been offered in the literature, first, with *Optimal Brain Damage*<sup>79</sup> which proposed in 1989 to prune network parameters based on their second derivatives, whereas in *Movement Pruning*<sup>117</sup>, the selection criteria is based on 1st-order gradient derivatives. Molchanov et al.<sup>92</sup> adopts a similar scheme via 1st-order Taylor expansion. These first-order pruning techniques are especially effective during model finetuning or transfer learning. In *Deep Compression*<sup>47</sup>, the pruning decision is based on the absolute value of the DNN parameter; it has been typically applied during the inference deployment. Probabilistic pruning techniques<sup>112,107</sup> have also emerged, showing high performance and promise.

Although these above-mentioned pruning methods can achieve very high sparsity levels, their main drawback is that they produce very irregular sparsity patterns that under-utilize the computing

hardware platform. To mitigate this problem, some hardware-friendly sparsity techniques have been put forward such as structured  $N:M$  pruning<sup>156,168</sup>, channel-wise<sup>171,160</sup>, and filter-wise<sup>50,87</sup> pruning.

**Quantization** seeks to reduce the memory requirements of deep learning models by representing the weights, activations, and gradients of the DNN using a smaller number of bits than their original encoding representation. This enables the model to run with greater throughput and higher energy efficiency, which is especially beneficial in embedded systems and edge devices. Quantization can be applied during inference deployment (known as post-training quantization), and/or during training (known as quantization-aware training). It also sometimes makes use of different bitwidths during the AI computation (known as mixed-precision).

Some notable work on this front includes outlier-aware quantization techniques such as OLAcel<sup>102</sup> and OCS<sup>167</sup>. Moons et al.<sup>93</sup> performs adaptive fixed-point quantization in the search for a minimum energy network. Stripes<sup>62</sup> employs bit-serial computing to enable configurable per-layer adaptive precision. Several quantization techniques such as Khoram et al.<sup>66</sup> aim to determine the appropriate per-layer bitwidth. Term quantization<sup>164,76</sup> operates on power-of-two terms to express DNN parameter values. It dynamically selects a fixed number of largest terms based on their relative rankings against terms of other values in the group. While 8-bit integer quantization (INT8)<sup>91</sup> has been already standardized in many ML computing platforms, some more aggressive techniques such as binary<sup>20</sup>, ternary<sup>170</sup>, and quaternary weight quantization<sup>17</sup> have been demonstrated during DNN inference with modest accuracy degradation. Sometimes, quantization is skipped on the sensitive first and last layers in order to escape steeper end-to-end accuracy loss. In contrast, in our work *AdaptivFloat* which is discussed in Chapter 3, rather than physically changing the bitwidth to adapt to a network dynamic range requirement, we seek to obtain the best numerical representation out of a chosen bitwidth regimen by dynamically adapting, per-layer, the floating-point exponent range.

### 2.3.2 PROMINENT HARDWARE ACCELERATORS AND SYSTEM-ON-CHIPS FOR AI AND NLP

Over the last decade, there has been tremendous research efforts focused on improving the performance and energy efficiency of AI hardware accelerators<sup>46,113,14,13,2,1,83,51,61,105,149,89,135,133,104,101,108,77,122,82,124,60</sup>. As these accelerators get deployed at all computing scales, there has been additional interest in the research community to automatically generate efficient designs<sup>150,148</sup>.

Some notable commercial AI accelerator architectures include NVIDIA server GPUs (e.g., V100, A100, H100) which are optimized for parallel processing of large-scale data. They consist of multiple streaming multiprocessors (SMs) that each contain several CUDA cores, which can execute multiple instructions concurrently<sup>98,99</sup>. Additionally, they utilize thread-level parallelism and memory-level parallelism to optimize performance.

Systolic arrays<sup>75</sup> have gained widespread usage in various commercial DNN accelerators due to their efficiency and utilization performance in carrying out large-scale matrix multiplications. Perhaps the most popular adaptation of a commercial accelerator based on the systolic architecture is the Google TPU<sup>61</sup>, which utilizes a  $256 \times 256$  systolic array in its matrix multiply unit (MMU).

Graphcore developed a highly parallel tiled-based architecture called IPU<sup>58,35</sup> for accelerating deep learning computations. Notably, their GC200 MK2 IPU processor stands out for provisioning one of the largest on-chip memory capacities for a commercial AI SoC, with up to 896MB of on-chip SRAM per chip.

**Specialized hardware for NLP.** Although the bulk of DNN hardware research has been targeted towards MLPs, CNNs, and RNNs, there have also been increased efforts to specialize the hardware architecture for Transformer-based NLP models<sup>68,64,154</sup>, which pose distinct challenges due to their network architectures and a computational cost that quadratically scales with the sequence length. Moreover, given large language models are currently driving a generative AI arms race, we can expect to see on the horizon commercialized hardware solutions specifically optimized

for Transformers' computation.

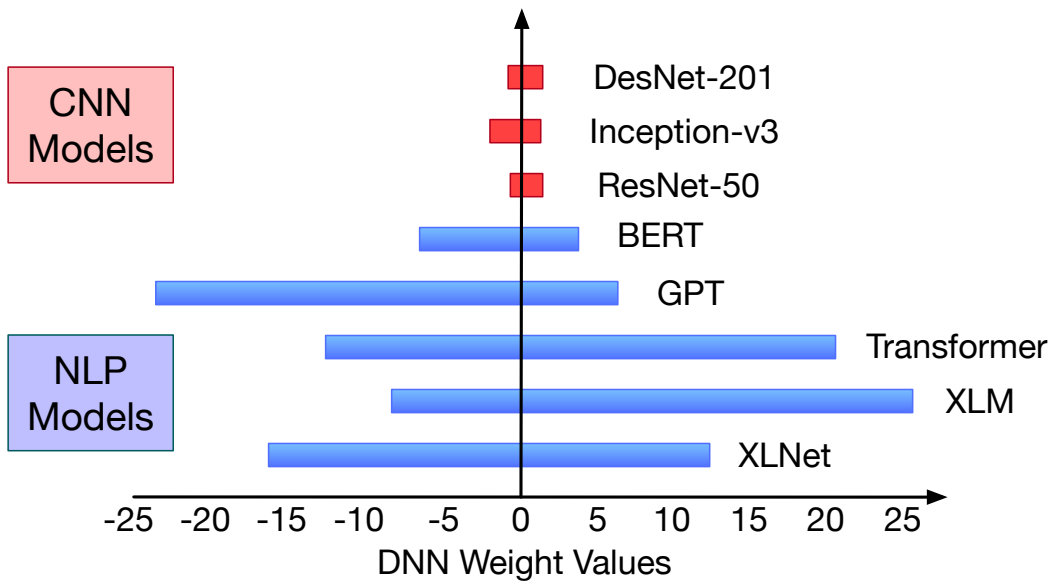
Notable work in this regard includes  $A^3$ <sup>44</sup>, which proposed a hardware architecture that reduces the number of computations in attention mechanisms via approximate and iterative candidate search. GOBO<sup>161</sup> focuses on BERT quantization via 3-bit clustering on the majority of BERT weights while storing the outlier weights and activations in full FP<sub>32</sub> precision. Although this scheme significantly reduces DRAM accesses, it requires a mixed-precision computational datapath and a non-uniform memory storage. OPTIMUS<sup>103</sup> accelerates Transformers with compressed sparse matrix multiplications and by skipping redundant decoding computations. SpAtten<sup>152</sup> accelerates Transformer-based models via progressive cascade token and attention head pruning. The importance of each attention head is determined during the computation via a top-k ranking system. DOTA<sup>109</sup> is a software-hardware co-design to detect and omit weak connections in attention graphs in order to skip the corresponding computations and memory accesses. ELSA<sup>45</sup> is a specialized hardware accelerator that exploits opportunities for approximation and parallelism in the self-attention operation to improve its performance and energy efficiency. It uses sign random projection to estimate the angle between query and key vectors. Sanger<sup>85</sup> proposes quantizing the query and key values before calculating the attention score. By doing so, insignificant entries in the attention score end up being zeroed out. LeOPArD<sup>81</sup> performs a bit-serial computation scheme during the query-key matrix multiplication in order to evaluate opportunities for early inference termination.

Section 5 describes our algorithm-hardware approach to accelerating Transformers by leveraging adaptive attention span, and entropy-controlled early exit and voltage-frequency scaling, which, notably enables the processor to optimize its latency and energy consumption at a finer-grained sentence-level granularity.

# 3

## Enabling Adaptive and Energy-Efficient Quantized Computations

DEEP LEARNING IS INHERENTLY RESILIENT TO LOW PRECISION BIT REPRESENTATION. Given this insight, AI hardware architects have been looking to achieve greater TOPS-per-Watt by increasing the on-chip arithmetic density. Consequently, a plethora of quantization techniques have been



**Figure 3.1:** Range of weights from popular CNN and NLP models. Weights in NLP models can be more than 10× larger than the maximum absolute weight value of common CNNs.

proposed to reduce the storage overheads of DNN parameters by converting them from native 32-bit floating-point types to significantly reduced bit widths. For example, quaternary<sup>17</sup>, ternary<sup>170</sup>, and binary<sup>20</sup> quantization techniques have been demonstrated. These fixed-point techniques are often assessed on CNNs or on shallow models that exhibit a relatively narrow weight distributions. However, as illustrated in Figure 3.1, sequence transduction NLP models, such as the Transformer<sup>147</sup>, feature weights that are more than an order of magnitude larger than those in popular CNN models such as ResNet-50, that use batch normalization. This is explained by the fact that vision CNN models tend to adopt batch normalization during training which effectively produces a weight normalization<sup>115</sup> side effect, whereas layer normalization<sup>7</sup> is characterized by invariance properties that do not reparameterize the network.

Furthermore, parameter spreads in NLP models tend to follow a Gaussian distribution whose center is widely skewed away from zero. For example, in GPT-1, the minimum and maximum values



are about -2.5 and 5, respectively, whereas the reverse spread is observed for XLM. Short of using 32-bit or 16-bit floating-point encodings, it is clear that a low precision number system (i.e.,  $\leq 8$ -bit) needs the ability to self-adapt to arbitrary parametric distributions, in a way that is not detrimental to the application’s accuracy. In essence, a high-performance, low precision data type for deep learning requires adaptive tensor scaling.

For this purpose, I proposed `AdaptiveFloat`<sup>134</sup>.

### 3.1 ADAPTIVFLOAT: AN ADAPTIVE FLOATING-POINT BASED NUMBER SYSTEM FOR ACCURATE AND RESILIENT DNN INFERENCE

In the pursuit of wider dynamic range and improved numerical accuracy, there has been surging interest in floating-point based<sup>30,120,73</sup>, logarithmic<sup>59,80</sup> and posit number systems<sup>43</sup>, which also form the inspiration of this work. Using a floating-point template, `AdaptiveFloat` improves on the aforementioned techniques by dynamically maximizing its available dynamic range at a neural network layer granularity.

The insight here is that DNN parameters of larger magnitude typically bear a disproportionate impact on inference accuracies compared to parameters of smaller magnitude. In fact, popular pruning strategies<sup>47</sup> recommend eliminating small weights that are close to zero in order to sparsify the neural network. Given this insight, `AdaptiveFloat` formulates and introduces in its floating-point exponent field, a variable bias,  $exp_{bias}$ , that is computed from the maximum absolute value in a given DNN tensor as shown in Equation 3.1. This, in effect, dynamically shifts the range of exponent values towards the maximum absolute value, enabling the latter to be represented as faithfully as possible for a given bit width regime.

$$-1^{sign} * mantissa * 2^{exponent+exp_{bias}} \tag{3.1}$$

---

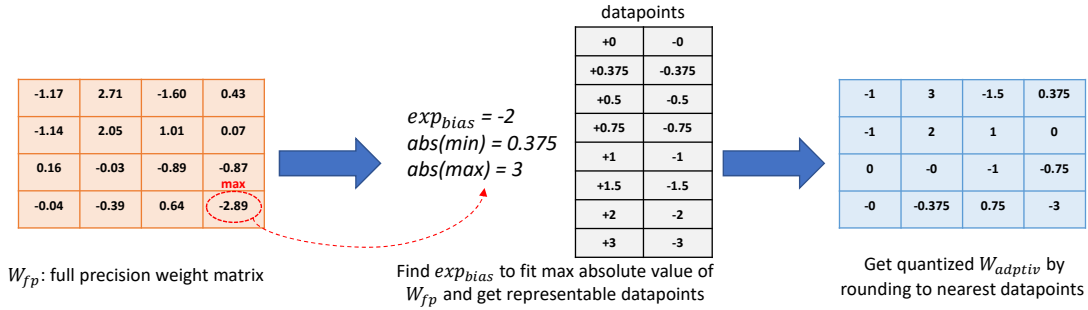
**Algorithm 1:** AdaptivFloat Quantization

---

**Input:** Matrix  $W_{fp}$ , bitwidth  $n$  and number of exponent bits  $e$   
// Get Mantissa bits  
 $m := n - e - 1$   
// Obtain sign and abs matrices  
 $W_{sign} := sign(W_{fp}); W_{abs} := abs(W_{fp})$   
// Determine  $exp_{bias}$  and range  
Find normalized  $exp_{max}$  for  $max(W_{abs})$  such that  
 $2^{exp_{max}} \leq max(W_{abs}) < 2^{exp_{max}+1}$   
 $exp_{bias} := exp_{max} - (2^e - 1)$   
 $value_{min} := 2^{exp_{bias}} * (1 + 2^{-m})$   
 $value_{max} := 2^{exp_{max}} * (2 - 2^{-m})$   
// Handle unrepresentable values  
Round  $value < value_{min}$  in  $W_{abs}$  to 0 or  $value_{min}$   
Clamp  $value > value_{max}$  in  $W_{abs}$  to  $value_{max}$   
// Quantize  $W_{fp}$   
Find normalized  $W_{exp}$  and  $W_{mant}$  such that  
 $W_{abs} = 2^{W_{exp}} * W_{mant}$ , and  $1 \leq W_{mant} < 2$   
 $W_q :=$  quantize and round  $W_{mant}$  by  $scale = 2^{-m}$   
// Reconstruct output matrix  
 $W_{adptiv} := W_{sign} * 2^{W_{exp}} * W_q$   
**return**  $W_{adptiv}$

---

Algorithm 1 describes how  $exp_{bias}$  is computed offline in order to quantize and scale the input tensor. First, the sign matrix,  $W_{sign}$ , and the matrix of absolute values,  $W_{abs}$ , are computed from the full precision weight matrix  $W_{fp}$ . Then, the algorithm finds the maximum absolute value from  $W_{abs}$  to determine the  $exp_{bias}$  corresponding to a suitable range of representable datapoints for the weight matrix for a given word and mantissa bit width regime. Prior to quantizing  $W_{abs}$ , the values smaller than the minimum representable normal value are rounded to zero or  $value_{min}$  at a halfway threshold. Then, values larger than the max value,  $value_{max}$ , are clamped  $value_{max}$ . The quantization involves rewriting  $W_{abs}$  into normalized exponent and mantissa form with an exponent matrix



**Figure 3.2:** Illustration of an AdaptiveFloat quantization (word width = 4, number of exponent bits = 2) from a full precision weight matrix

$W_{exp}$  and a mantissa matrix  $W_{mant}$ . The mantissa matrix is quantized by the quantization scale calculated by the number of mantissa bits. Figure 3.2 provides a graphical illustration of the AdaptiveFloat quantization for a 4-bit word regime along with a 2-bit exponent field.

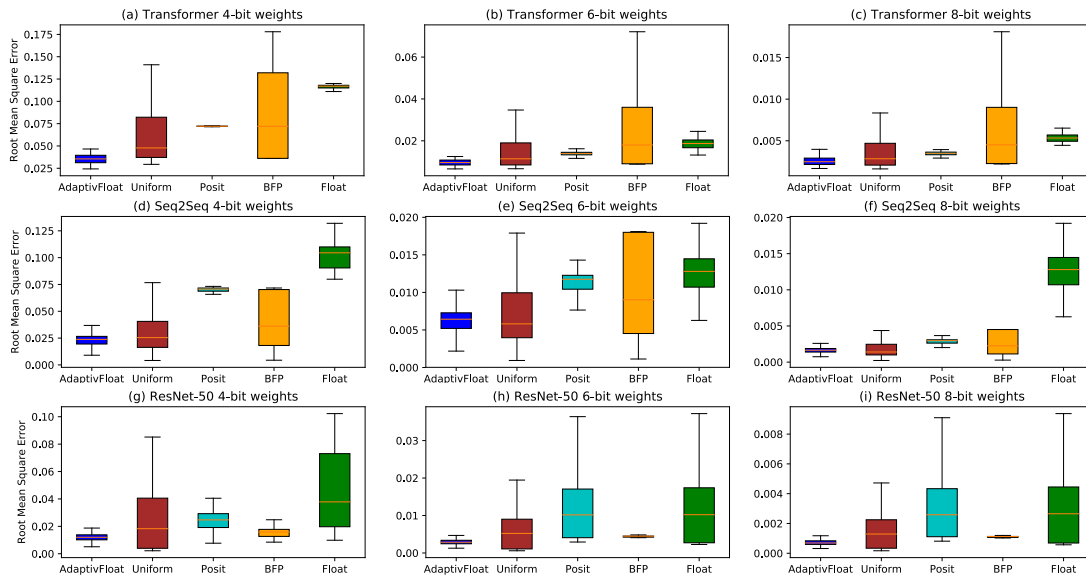
We note that AdaptiveFloat breaks from IEEE-754 standard due to its unique clamping strategy and the fact that there is no support for denormal numbers. This framework enables us to engineer leaner hardware as discussed later in Section 3.3.

### 3.2 EXPERIMENTAL RESULTS

We evaluate bit compression performance on three widely used DNN models of distinct neural types that demonstrate a wide range of weight distributions. As shown in Table 3.1, they include: (1) Transformer, which has had a significant impact on machine translation and question answering; (2) a 4-layer LSTM encoder, 1-layer LSTM decoder, attention-based sequence-to-sequence (Seq2Seq) network frequently utilized in speech recognition; and (3) ResNet-50, a well-known image classification CNN<sup>49</sup>. The Transformer and Seq2Seq networks were trained on the OpenNMT platform<sup>70</sup> using the WMT’17 English-to-German and LibriSpeech datasets, respectively. And, ResNet-50 was trained on the Pytorch framework using the ImageNet dataset. The representation

**Table 3.1:** DNN models under evaluation

MODEL	APPLICATION	DATASET	STRUCTURE	NUMBER OF PARAMETERS	RANGE OF WEIGHTS	FP <sub>32</sub> PERFORMANCE
TRANSFORMER	MACHINE TRANSLATION	WMT'17 EN-TO-DE	ATTENTION, FC LAYERS	93M	$[-12.46, 20.41]$	BLEU: 27.40
Seq2Seq RNNs	SPEECH-TO-TEXT	LIBRISPEECH 960H	ATTENTION, LSTM, FC LAYERS	20M	$[-2.21, 2.39]$	WER: 13.34
RESNET-50	IMAGE CLASSIFICATION	IMAGENET	CNN, FC LAYERS	25M	$[-0.78, 1.32]$	TOP-1 ACC: 76.2



**Figure 3.3:** Root mean square of the quantization error w.r.t. FP<sub>32</sub> at 4-bit, 6-bit and 8-bit weight precision across the layers of the Transformer, Seq2Seq and ResNet-50 models. AdaptiveFloat produces the lowest mean error compared to the other number systems.

performance of AdaptiveFloat is compared with numerical data types commonly employed for deep learning computations, namely block floating-point (BFP), IEEE-like float, posit, and uniform (or integer) representations.

We note here that AdaptiveFloat, integer, and BFP formats are self-adaptive in the sense that their dynamic ranges auto-adjust, layer-by-layer, based on a scale factor computed from the parametric distribution in the tensor. For example, the integer format uses a floating-point scale factor computed from the maximum value in the tensor in order to normalize its range. And, BFP utilizes a

**Table 3.2:** Impact of weight bit compression post-training quantization / post-quantization aware retraining

#BITS	BLEU SCORE OF TRANSFORMER (BLEU @ FP32=27.4)					WORD ERROR RATE OF SEQ2SEQ (WER @ FP32=13.34)					TOP-1 ACCURACY OF RESNET-50 (TOP-1 ACC. @ FP32=76.2)					
	FLOAT	BFP	UNIFORM	POSIT	ADAPTIVFLOAT	FLOAT	BFP	UNIFORM	POSIT	ADAPTIVFLOAT	FLOAT	BFP	UNIFORM	POSIT	ADAPTIVFLOAT	
16	27.4 / 27.4	27.4 / 27.4	27.4 / 27.4	27.4 / 27.4	27.4 / 27.5	27.4 / 27.6	13.40 / 13.07	13.30 / 13.14	13.27 / 12.82	13.29 / 13.05	13.27 / 12.93	76.1 / 76.3	76.2 / 76.3	76.1 / 76.3	76.1 / 76.3	76.2 / 76.3
8	27.2 / 27.5	26.3 / 27.3	27.3 / 27.4	27.3 / 27.5	27.3 / 27.7		14.06 / 12.74	13.23 / 13.01	13.28 / 12.89	13.24 / 12.88	13.11 / 12.59	75.4 / 75.9	75.7 / 76.0	75.9 / 76.1	75.4 / 76.0	75.7 / 76.3
7	27.1 / 27.5	16.9 / 26.8	26.0 / 27.2	27.3 / 27.4	27.3 / 27.7		13.95 / 12.84	13.54 / 13.27	13.45 / 13.37	13.36 / 12.74	13.19 / 12.80	73.8 / 75.6	74.6 / 75.9	75.3 / 75.9	74.1 / 75.8	75.6 / 76.1
6	26.5 / 27.1	0.16 / 8.4	0.9 / 23.5	26.7 / 27.2	27.2 / 27.6		15.53 / 13.48	14.72 / 14.74	14.05 / 13.90	15.13 / 13.88	13.19 / 12.93	65.7 / 74.8	66.9 / 74.9	72.9 / 75.2	68.8 / 75.0	73.9 / 75.9
5	24.2 / 25.6	0.0 / 0.0	0.0 / 0.0	25.8 / 26.6	26.4 / 27.3		20.86 / 19.63	21.28 / 21.18	16.53 / 16.25	19.65 / 19.13	15.027 / 12.78	16.1 / 73.6	13.2 / 73.4	15.1 / 74.0	33.0 / 73.9	67.2 / 75.6
4	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	16.3 / 25.5		INF / INF	76.05 / 75.65	44.55 / 45.99	INF / INF	19.82 / 15.84	0.5 / 66.3	0.5 / 66.1	2.6 / 67.4	0.7 / 66.7	29.0 / 75.1

shared exponent which is also derived from the maximum value in the tensor or neural network layer. Therefore, the concept of a scale factor for lateral shift of numerical distributions is represented by the exponent bias,  $exp_{bias}$ , in the case of AdaptivFloat, and it is the floating-point scaling factor in the case of integer formats, and the shared exponent in the case of BFP data types. In this evaluation, Float and Posit formats do not use scale factors.

Figure 3.3 show the distribution of the root mean squared (RMS) quantization error emanating from the data types and computed across the different layers of the three models under evaluation. We can observe that AdaptivFloat consistently produces lower average quantization error compared to uniform, BFP, posit, or IEEE-like float encoding. Furthermore, among the self-adaptive data types, the boxplots show that AdaptivFloat exhibits the narrowest error spread of all the evaluated precisions.

Table 3.2 shows the accuracy of the data types under study as they are put to the test under varying weight bit precisions on the Transformer, sequence-to-sequence, and ResNet-50 models. The inference results are tabulated during post-training quantization (PTQ) and after quantization-aware training (QAT) from the plateaued FP32 baseline. The key observation we can distill is that AdaptivFloat demonstrates much greater robustness at very low precision ( $\leq 6$ -bit) compared to the other four data formats. Notably, at 4-bit encoding, AdaptivFloat can still deliver a respectable BLEU score of 25.5 on the Transformer model following retraining. In contrast, the other four number formats have a catastrophic effect due to insufficient dynamic range. Similar findings ap-

**Table 3.3:** Impact of both weight and activation quantization, measured after quantization-aware retraining

#BITS	BLEU SCORE OF TRANSFORMER (BLEU @ FP32=27.4)					WORD ERROR RATE OF SEQ2SEQ (WER @ FP32=13.34)					TOP-1 ACCURACY OF RESNET-50 (TOP-1 ACC. @ FP32=76.2)				
	FLOAT	BFP	UNIFORM	POSIT	ADAPTIVFLOAT	FLOAT	BFP	UNIFORM	POSIT	ADAPTIVFLOAT	FLOAT	BFP	UNIFORM	POSIT	ADAPTIVFLOAT
W8/A8	27.4	27.4	10.1	26.9	27.5	12.77	12.86	12.86	12.96	12.59	75.7	75.7	75.9	75.8	76.0
W6/A6	25.9	0.0	5.7	25.7	27.1	14.58	14.68	14.04	14.50	12.79	73.5	73.4	74.1	73.6	75.0
W4/A4	0.0	0.0	0.0	0.0	0.3	INF	78.68	48.86	INF	21.94	63.3	63.0	64.3	63.0	72.4

ply to the seq2seq and ResNet-50 models, as AdaptivFloat demonstrates only a modest decrease in performance after retraining with 4-bit and 5-bit weight precision. For instance, only a 1.2 Top-1 accuracy drop is seen with a weight width of 4-bit. When the weights of the seq2seq model are quantized to 4-bit, the non-adaptive data types (Float and Posit) are essentially unable to provide expressible transcription. This suggests that, for robust performance at low precision, **it is critical to have a quantization scheme that can adjust its available dynamic range to encode the compressed weights as accurately as possible.** AdaptivFloat’s exceptional robustness at extremely low precision allows for greater compute density in reconfigurable architectures with a comparatively minor loss in computational accuracy.

Moreover, research has demonstrated that incorporating noise into weight values during gradient computations can provide a regularization benefit, leading to enhanced generalization performance<sup>96</sup>. This improvement can be observed across all number formats, but is especially noticeable in AdaptivFloat, which has been found to outperform FP32 in terms of BLEU score by as much as +0.3, reduce word error rate by -0.75, and increase Top-1 accuracy by +0.1.

Tables 3.3 shows the inference performance from quantizing both weights and activations.  $W_n/A_n$  signifies a quantization of  $n$ -bit weight and  $n$ -bit activation. Our findings demonstrate that AdaptivFloat’s 8-bit performance is comparable to, if not better than, the baseline FP32 results for all three DNN models. And even the degradation observed at 6-bit remains relatively minimal. Interestingly, in the case of the seq2seq model, 6-bit AdaptivFloat weight and activation quantization generates sufficient regularization to surpass the FP32 baseline. However, when we reduce the

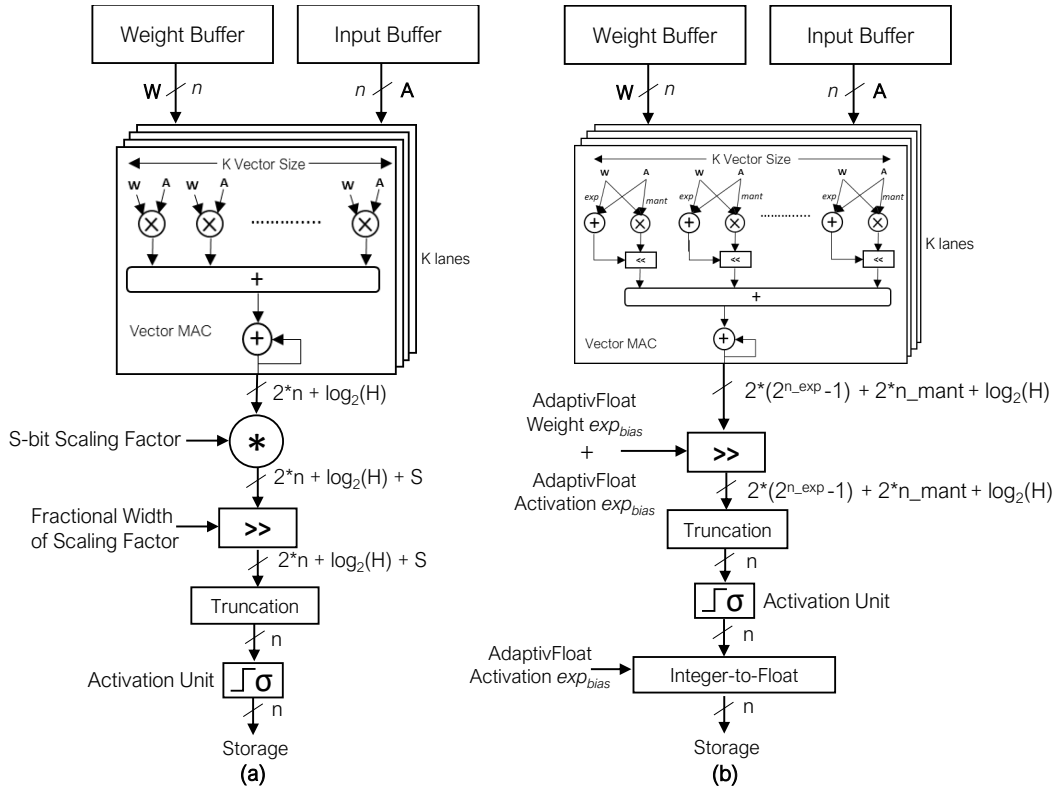


Figure 3.4: (a) NVDLA-like  $n$ -bit Integer-based PE<sup>150</sup>, (b) Proposed  $n$ -bit Hybrid Float-Integer PE.

weight and activation precision to 4 bits, the performance degradation in AdaptiveFloat is more substantial for the sequence models compared to ResNet-50. This is because many activations from the attention mechanism start falling outside of the available dynamic range of the number format.

AdaptiveFloat’s superior bit compression ability paves the way to efficient bit packing into resource-constrained accelerators. In the next section, we describe how we translate the AdaptiveFloat algorithm in the design of a floating-point multiply-and-accumulate (MAC) datapath.

### 3.3 HARDWARE IMPLEMENTATION

To maximize the benefits of AdaptivFloat bit packing ability, we co-designed a Hybrid Float-Integer (HFINT) Processing Element (PE), shown in Figure 3.4(b), that exploits AdaptivFloat logic in its computational datapath and strikes an efficient balance between the high accuracy of floating-point computations and the greater hardware density of fixed-point post-processing. We compare the proposed HFINT PE architecture with a NVDLA-like PE, shown in Figure 3.4(a), that uses monolithic integer arithmetic, which is commonly found in many commercial ML accelerators (e.g., TPU<sup>61</sup>, Ethos-U55<sup>6</sup>).

In the HFINT PE, the vector MAC units perform floating-point multiplications between a  $n$ -bit float weight vector and a  $n$ -bit float activation vector, and then accumulates the result in a fixed-point format. The per-layer AdaptivFloat  $exp_{bias}$  values for weights and activations are stored in 4-bit registers, and are used to shift the exponent range of the accumulated partial sums during on-chip inference. We note that while the AdaptivFloat  $exp_{bias}$  for the static weights are extracted post-training, the  $exp_{bias}$  for the dynamic activations are informed from statistics during offline batch inference on the test dataset. The accumulation precision is  $2 * (2^{n_{exp}} - 1) + 2 * n_{mant} + \log_2(H)$ -bit in order to accumulate up to  $H$  values without overflow. The accumulated partial sums are then clipped and truncated back to  $n$ -bit integer before being processed by the activation function. At the end of the datapath, the integer activations are converted back to the AdaptivFloat format for compact storage into the accelerator scratchpads.

In contrast, the INT PE (Figure 3.4(a)) utilizes a wider multiplier in the MAC unit to process a pair of fixed-point operands. This datapath, notably, requires a post-accumulation multiplier for adaptive scaling of the partial sums. In the next section, we provide energy, performance, and area comparisons between the two PE topologies.



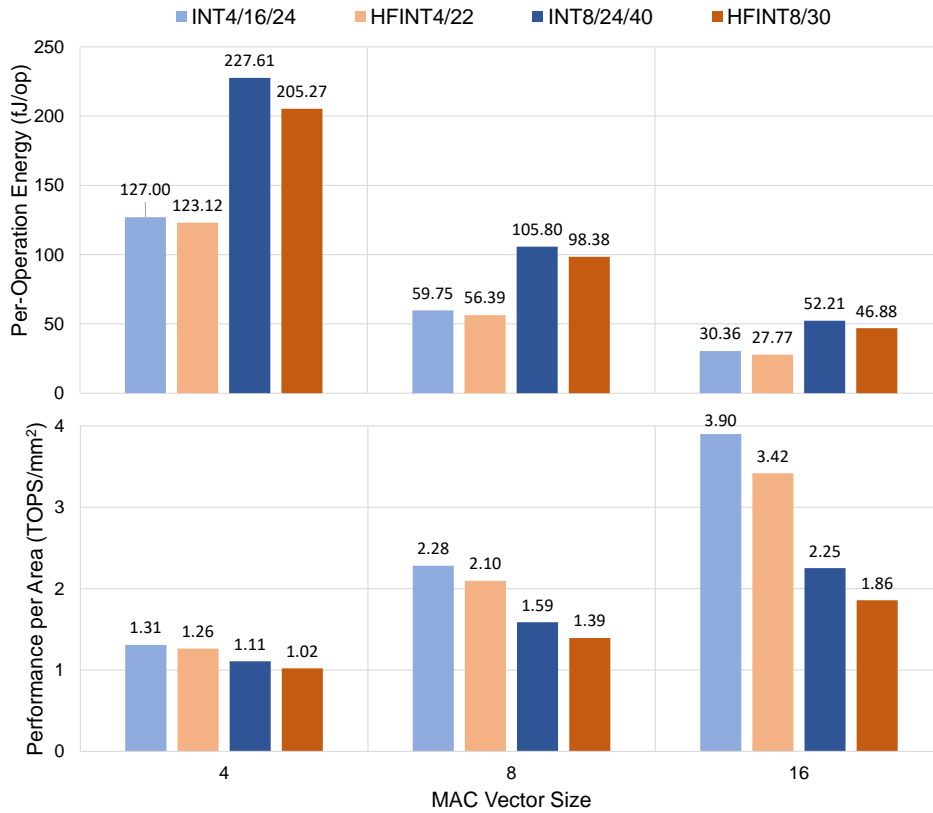
### 3.4 RESULTS AND IMPLICATIONS

We designed the INT and HFINT processing elements in SystemC while also leveraging synthesizable components from the MatchLib library<sup>65</sup>. The Verilog RTL was autogenerated by the Catapult high-level synthesis tool with HLS constraints uniformly set with the goal to achieve maximum throughput on the pipelined designs. The two designs employ the same evaluation methodology. Energy, performance, and area results are reported on the post-HLS Verilog netlists by the Catapult tool at 1GHz clock frequency using a commercial 16nm FinFET standard cell library. The simulated workload consists of 100 LSTM time steps with 256 hidden units operating in a weight stationary dataflow. Within each PE, there is an input and bias buffer that ranges from 1KB to 4KB in size, as well as a weight buffer whose size ranges from 256KB to 1MB, depending on the vector size and operand bit width. Also, in this experimental evaluation, the number of exponent bits inside the HFINT MAC is fixed to 3.

Figure 3.5 evaluates the effect of increasing throughput via the MAC vector size,  $K$ , which also equals the number of parallel MAC lanes, meaning that a single PE throughput equals  $K^2 10^9 OPS$ . The top row shows the energy efficiency in terms of joule per operation while the bottom row shows performance per area in terms of TOPS per  $mm^2$ .

We observe that larger vector sizes and operand bit widths benefit more the HFINT PE than the INT PE in terms of energy efficiency. The smaller per-operation energy of the HFINT PE stems from the fact that its vector MACs contain smaller mantissa multipliers and exponent adders that consume less overall power than the full bitwidth multipliers used in the INT PE MACs. This advantage grows larger as the MAC vector size increases due to higher spatial reuse of the accumulated partial sums. On the other hand, the INT PEs exhibit  $1.04 \times$  to  $1.21 \times$  higher performance per unit area compared to the HFINT PEs due to its more compact and homogeneous logic.

Finally, we note that the two PE topologies achieve the same end-to-end compute time (or time-



**Figure 3.5:** Per-operation energy (**Top**) and throughput per unit area (**Bottom**) of the INT and HFINT PEs across MAC vector sizes.

to-solution) due to incurring the same aggregate pipelining budget.

The next section discusses the integration of AdaptivFloat in a design-space-exploration (DSE) framework which optimizes the search and the configuration of the data type for efficient accelerator co-design.

### 3.5 FACILITATING DATA TYPE EXPLORATION FOR EFFICIENT HARDWARE DESIGN

The exploration of newer number formats in the context of deep learning (DL) and other emerging applications is challenging, primarily since the majority of compute fabrics available today (namely,

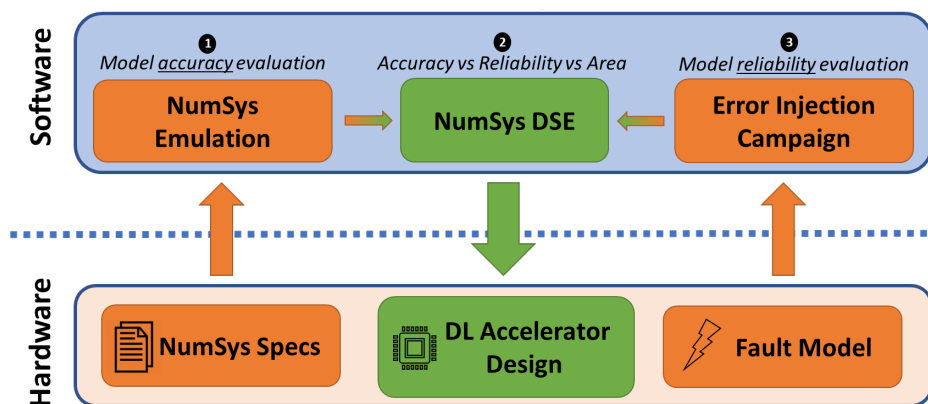


Figure 3.6: GoldenEye's utility within a co-designed ML and hardware ecosystem.

CPU and GPU) supports a limited set of number formats. Consequently, programming interfaces (e.g., CUDA, OpenCL) and DL frameworks (e.g., PyTorch, TensorFlow) are forced to optimize along restrictive dimensions in the space of possible number formats. To address this challenge, we proposed GoldenEye<sup>88</sup>, an open-source number format simulator that enables the selection of configurable number systems as a first-order parameter when evaluating their impact on DNN classification accuracy.

As illustrated in Figure 3.6, GoldenEye provides a platform for fast number system emulation in software, allowing users to capture a model's accuracy as a function of the underlying hardware representation. The number formats that can be explored include floating-point and block floating-point numerics, integer, radix-based fixed-point, and AdaptiveFloat. Together, along with a heuristic for domain search and bit width precision exploration, the tool can inform future accelerator designs in the context of accuracy, reliability, and area, based on the numerical data formats explored.

My research work also developed a synthesizable library of multiply-and-accumulate (MAC) units<sup>136</sup> that allow the number format and its precision to be optimized during the precision-aware design space exploration of coarse-grained reconfigurable accelerators (CGRAs).

### 3.6 TAKEAWAYS

AdaptivFloat is a co-design between the application and the number system that informs a generalized floating-point based mathematical blueprint for adaptive and resilient DNN quantization. It can be easily applied on neural models of various categories (CNN, RNN, MLP, Transformers), layer depths and parameter statistics. AdaptivFloat formulates and introduces a **configurable exponent bias**, that is computed from the maximum absolute value in a given neural network layer. This allows the range of representable values to be dynamically shifted in order to accurately encode parameters of larger magnitude, as they bear a disproportionate impact on inference accuracy.

### 3.7 FOLLOW-ON RESEARCH

In this work,  $exp_{bias}$  is computed at a neural network layer granularity. For greater computational accuracy at especially lower bit precision ( $\leq 6$ -bit), one may further decrease the space wherein the scaling factor is computed. For example, per-vector scaled integer quantization<sup>23,64</sup> has been proposed whereby the scaling factor for the integer quantization is computed at a hardware vector granularity. To reduce hardware overheads, the scaling factor may also be calculated within a coarser or more structured search space, such as, at a neural network channel level. In Chapter 5, we discuss a 4-bit floating-point MAC for Transformer computations where  $exp_{bias}$  is computed within a 32-element hardware vector space.

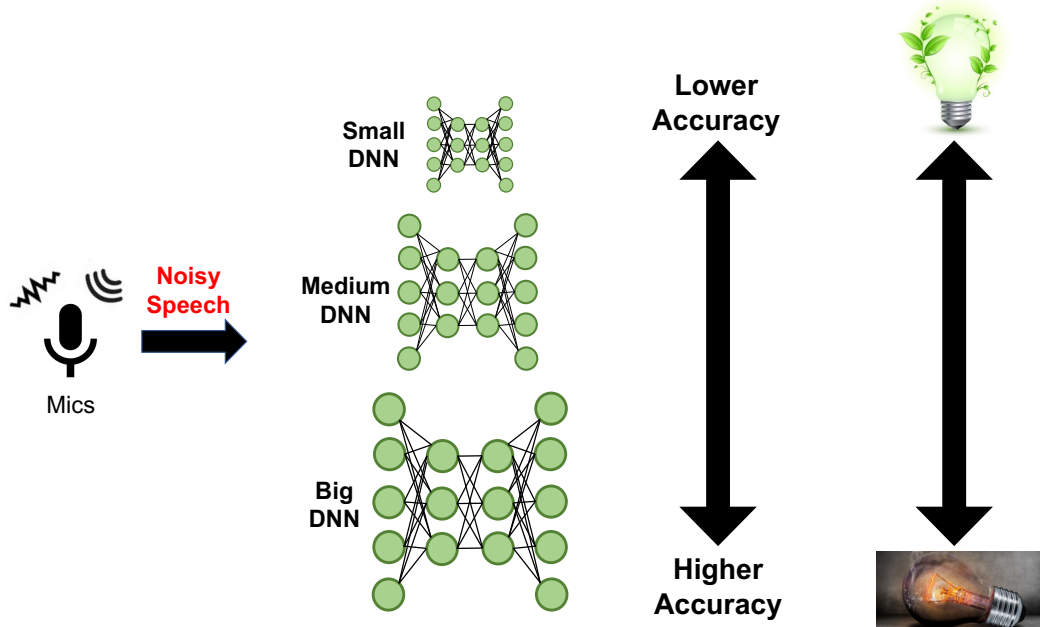
Another natural extension of this work would be to compute the activations'  $exp_{bias}$  dynamically in the hardware, as opposed to offline. Given the weights are static during inference, it makes sense to compute their scaling factor offline. However, for greater computation accuracy, it would be desirable to dynamically calculate the scaling factor for the activations, which, unlike the weights, are dynamic in nature. The challenge, here, is to compute this activation scaling factor in as few cycles

as possible in order to avoid undue latency overheads. This is not a trivial undertaking as finding the maximum value in a tensor, which is required to compute  $exp_{bias}$ , may take multiple clock cycles. A sampling method to dynamically generate  $exp_{bias}$  in the hardware datapath, may offer a more comfortable and efficient compromise.

# 4

## Accelerating Edge AI Attention-based Speech-to-Text RNNs

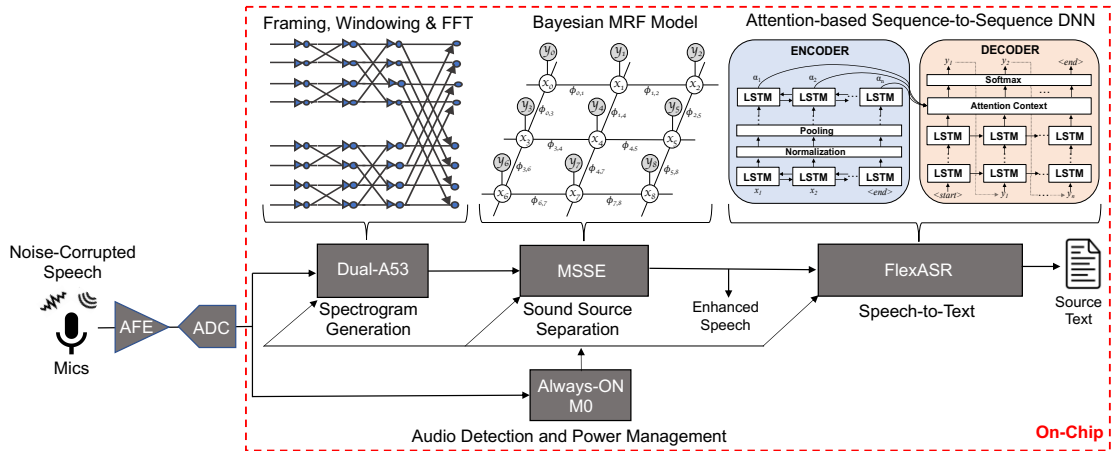
HOW TO ARCHITECT A FULLY-ON-CHIP NOISE-RESISTANT AUTOMATIC SPEECH RECOGNITION SYSTEM WITHOUT OVERSIZING THE DNN MODEL? In the tradeoff illustrated in Figure 4.1, we can buy noise resistance and high inference accuracy by increasing the size and complexity of the



**Figure 4.1:** Tradeoff between speech-to-text DNN model size and system energy efficiency for a speech-based conversational AI interface.

speech-to-text model<sup>\*</sup>. However, this leads to a very energy-inefficient solution with excess end-to-end latencies. Conversely, we can employ a smaller automatic speech recognition (ASR) DNN which exacts a much lower energy cost. However, poor transcription accuracy is observed given the leaner DNN is not as resilient against noise interference. We investigate how we may engineer the best possible compromise in a cost effective performance, power, and area envelope; and preferably, in a single monolithic system without integrating an analog noise-cancelling frontend chip<sup>94,118</sup> often used in IoT and wearable devices. Furthermore, from our point of view, as speech-based conversational AI interfaces are being applied to keyword spotting (KWS), ASR, natural language processing (NLP), and text-to-speech (TTS) applications, it is of paramount importance that they provide uncompromising performance for context learning in long sequences, and, that they work seam-

<sup>\*</sup>In this thesis, the terms automatic speech recognition and speech-to-text are used interchangeably.



**Figure 4.2:** Inference pipeline executed on the SM6 chip. Upon detecting audio, the M0 subsystem wakes up the accelerators and the A53 which produces spectrogram features that get denoised by the MSSE engine. Then, from the enhanced speech, FlexASR accelerates attention-based ASR workloads.

lessly in polyphonic environments. Furthermore, to provide the best possible user experience, it is desirable to achieve real-time throughput. This means the per-frame end-to-end latency should be lower than the frame length of the post-FFT spectrogram.

For this purpose, we present a co-design between the application and the system-on-chip (SoC) architecture to create a more efficient and noise-resistant speech processing solution which leverages some of the latest advances in deep learning and unsupervised machine learning. The proposed 25 mm<sup>2</sup> system-on-chip<sup>132,133</sup>, codenamed SM6 and developed in 16nm FinFET process, executes end-to-end speech-enhancing attention-based ASR and NLP workloads as shown in Figure 4.2. This hardware–software co-design framework demonstrates a tight coupling between special function accelerators and CPU processing. The SoC, notably, includes:

1. **FlexASR**, a highly reconfigurable NLP inference processor optimized for whole-model acceleration of bidirectional attention-based sequence-to-sequence (seq2seq) recurrent neural networks (RNNs). We further note that FlexASR processing elements support the AdaptiveFloat mechanism in their multiply-and-accumulate datapaths.



2. **MSSE**, which stands for Markov random field Source Separation Engine, is a probabilistic graphical model accelerator for unsupervised inference via Gibbs sampling. It is used for real-time sound source separation in this work.
3. **A dual-core Arm Cortex A53 CPU cluster**, which provides on-demand single Instruction/multiple data (SIMD) fast fourier transform (FFT) and feature extraction processing. The A53 cores also perform various application logic such as the expectation–maximization (EM) algorithm, and 8-bit floating-point (FP8) quantization.
4. **An always-ON Mo subsystem** for audio detection and power management.

We will show that by implementing Bayesian denoising prior to the speech recognition operation on the same chip, MSSE allows FlexASR to store a much smaller ASR model inside its scratchpad memories, which obviates the very inefficient strategy of scaling up the DNN model in order to achieve noise robustness. We note this is done without compromising the application accuracy.

#### 4.1 THE SM6 SoC ARCHITECTURE

In this section, we provide greater details on the SM6 SoC architecture, as well as the hardware–software co-design and verification methodology employed during the test chip implementation.

Figure 4.3 shows the overall SoC architecture comprising FlexASR, MSSE, an Arm Cortex-Mo microcontroller with a 128KB instruction SRAM, and a dual-core Arm Cortex-A53 CPU cluster with 2MB L2\$. (common in high performance embedded and mobile SoCs) connected together via 32-bit AHB and 128-bit AXI buses. The two accelerators (FlexASR and MSSE) are equipped with AXI-slave memory-mapped interfaces responding to AXI-master requests from the A53 and Mo CPUs. FlexASR interrupt signal (IRQ) is received by the A53 Generic Interrupt Controller (GIC) in order to facilitate the sequencing of instructions and computations between the A53 and FlexASR.

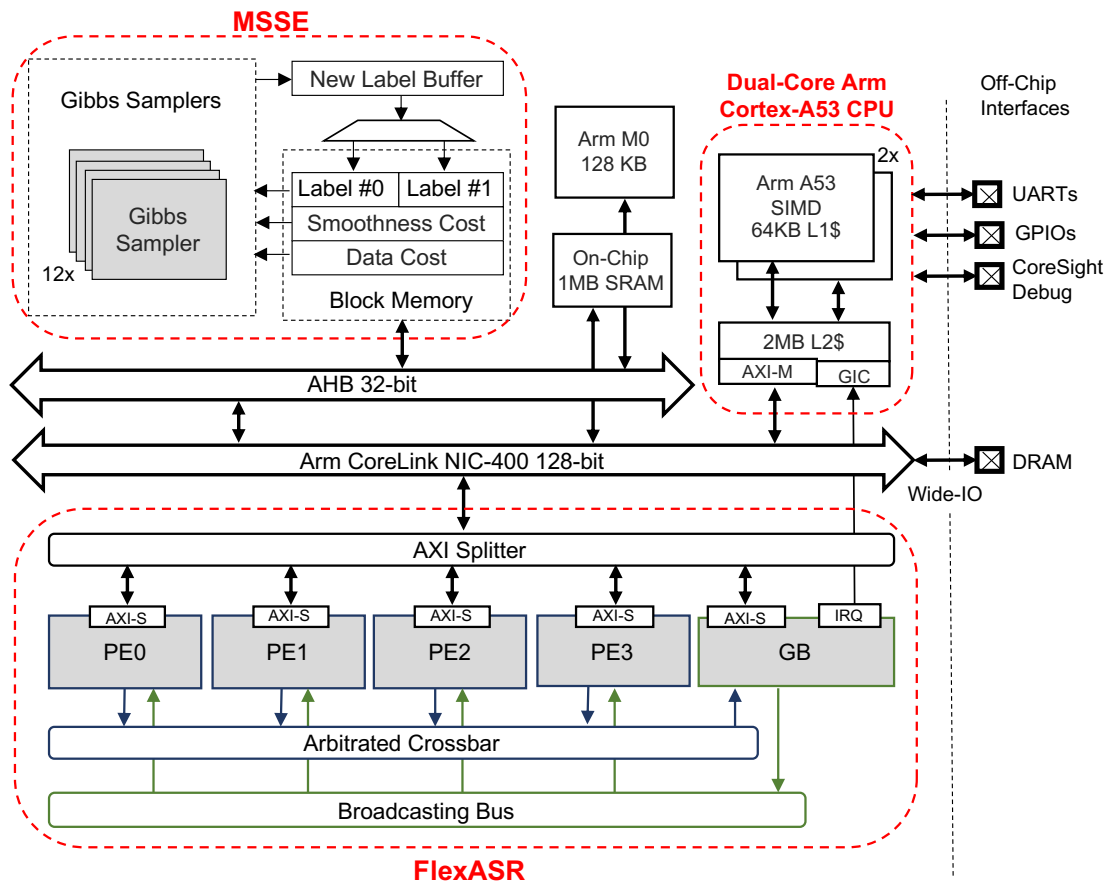


Figure 4.3: Block diagram of the SM6 SoC, highlighting main components.

A 1-MB SRAM buffer is provisioned at the top level in order to store the intermediate pre- and post-processed data of the inference pipeline shown in Figure 4.2. The SoC is also equipped with various off-chip interfaces required for DRAM access via field-programmable gate array (FPGA) and for debug.

#### 4.1.1 FLEXASR

FlexASR is designed to support the key computational kernels and features seen in LAS seq2seq networks (Fig. 2.2) as described in Table 4.1, while also allowing spatial and temporal configurations. As illustrated in Fig. 4.3, FlexASR consists of four processing elements (PEs) and a multi-

**Table 4.1:** Computations accelerated in FlexASR

Neural Transformation	Vanilla RNN	GRU	LSTM	Linear
RNN Sequence	Forward-Only		Bidirectional	
Pooling	Mean		Maximum	
Normalization	Layer Normalization			
Attention	Attention Mechanism <sup>8</sup>			

function global buffer (GB) unit.

The communication between GB and PEs is performed via custom-built channel links. A centralized arbiter is used to referee the stream of PE partial results which will be aggregated by the GB. Once the full activation has been collected, the GB will then broadcast it back to the PEs for the next time step computation. Each PE and GB is interfaced with an AXI-Slave port. An interrupt (IRQ) channel originating from the GB to the A53 cluster is implemented to indicate successful completion of the instructed task.

Section 4.2 further expands on the FlexASR architecture, detailing its PE, GB, and tiling mechanisms.

#### 4.1.2 MRF SOUND SOURCE SEPARATION ENGINE

Over the last decade, there has been extensive research on the design of ML accelerators to solve supervised learning problems. In contrast, unsupervised Bayesian models can be effective in solving problems relying on unlabeled data expressing various degrees of information uncertainty<sup>71,72</sup>. Unfortunately, Bayesian inference workloads do not efficiently scale on traditional CPUs and GPUs, therefore requiring specialized hardware.

In this work, we accelerate Gibbs sampling operations on Markov Random Fields (MRFs) for the purpose of denoising noise-corrupted speech or enhancing a particular acoustic source in an environment with multiple sound sources. The unsupervised Bayesian algorithm excels in a more dynamic environment such as when sources are moving with respect to the microphones<sup>67</sup>, which

can potentially create problematic corner cases for supervised methods where it is necessary to cover all scenarios with training data.

#### 4.1.3 DUAL-CORE ARM CORTEX-A53

The inference of the speech enhancing pipeline (Fig. 4.2) effects numerous dynamic data exchanges between the CPU and the accelerators. SM6 integrates two A53 CPU cores<sup>139</sup>, proven in high performance embedded and mobile SoCs, for the following versatile purposes:

1. Feature extraction tasks. Framing, windowing, and 1024-pt FFT tasks, required to synthesize the overlapping sequence of speech spectrograms, are vectorized using NE10 SIMD instructions<sup>142</sup>.
2. Accelerator programming. The AXI-Master port of the A53 issues ISA instructions to FlexASR and MSSE to configure the nature, shape, and size of their workloads.
3. Expectation-maximization (EM) algorithm which is a supplemental step of the Gibbs sampling process during sound source separation<sup>114</sup>.
4. 8-bit floating-point (FP8) quantization. As FlexASR PEs work on FP8 operands, the 32-bit fixed-point outputs from MSSE need to be converted and scaled down to lower bit precision.
5. Label mask convolution. The A53 convolves the binary label mask from MSSE with the original spectrogram in order to extract the *clean* speech.
6. Other miscellaneous tasks which include IRQ handling and operation system support.

#### 4.1.4 DESIGN AND VERIFICATION METHODOLOGY

In order to develop the SoC in an agile manner while minimizing tapeout risks, we leveraged the *CHIPKIT* SoC scaffold<sup>157</sup>, as well as, various ARM collaterals (e.g., A53 and Mo soft IPs, Arm

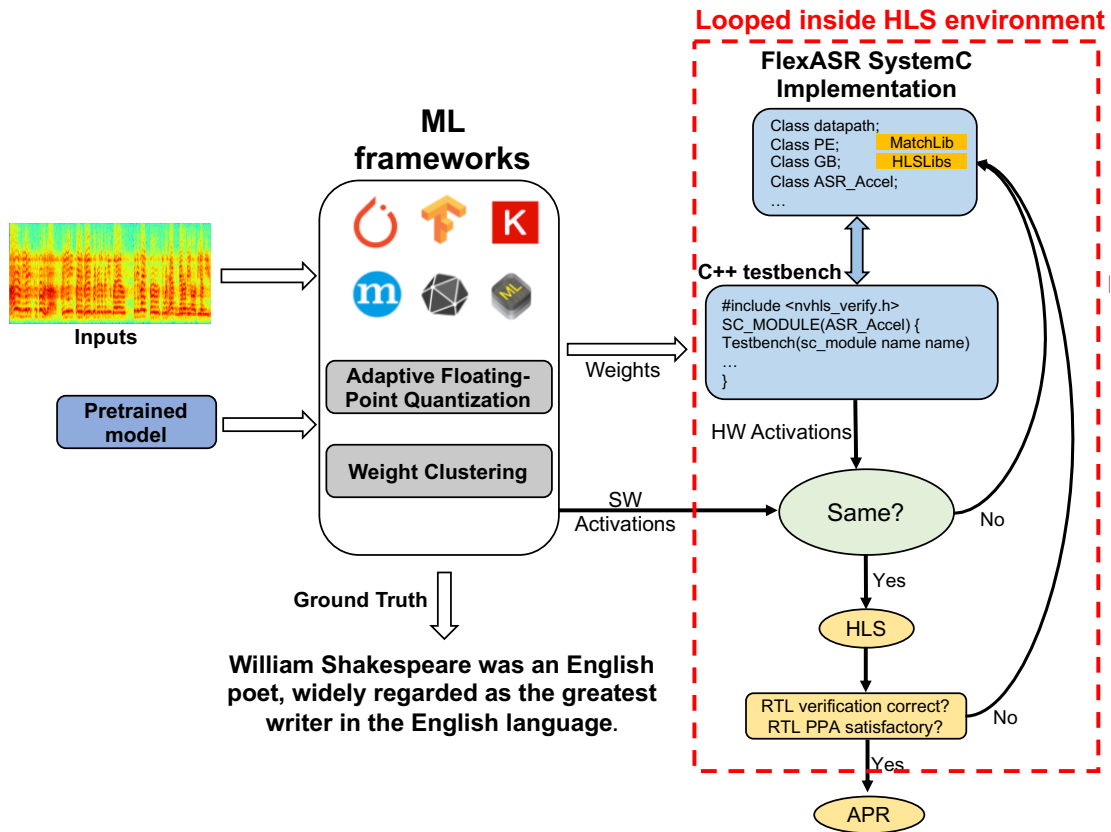


Figure 4.4: Algorithm-hardware co-design and verification methodology of the FlexASR accelerator.

Socrates for generating the NIC-400 interconnect), which allowed us to focus on the main differentiating features of the SoC. One such differentiation is in the hardware-software codesign of the FlexASR processor.

FlexASR was designed using object-oriented high-level synthesis (HLS) for fast SystemC-to-RTL prototyping<sup>65</sup>. In order to evaluate the bit-level correctness of the hardware on a realistic speech-to-text workload, we developed a design and verification flow, shown in Figure 4.4, which closes the loop between the software modeling and the backend hardware implementation being abstracted within the HLS environment. Considering the software ML framework (e.g., PyTorch, TensorFlow) to be golden, the HLS environment allowed us to quickly make hardware tweaks and ECOs

until *i*) the hardware and software DNN activations returned matching numerical results, *ii*) the Post-HLS verification is functionally correct, and *iii*) post-HLS PPA results are satisfactory. This agility is made possible by the higher level of abstraction imposed by the HLS flow.

The SystemC source code description of FlexASR is now publicly available <sup>140</sup>.

## 4.2 THE FLEXASR HARDWARE ACCELERATOR

In this section, we provide details of the FlexASR architecture to accelerate LAS models illustrated in Figure 2.2. We can categorize the seq2seq computations into two main parts: (1) RNN computations for each time step, which mainly involve matrix-vector multiplications (MVMs) with fixed weights and dynamic activations, and (2) auxiliary operations like attention, normalization, and pooling, which involve activations across time steps.

For the first case, four processing elements (PEs) with 16 lanes of vector MACs are provisioned to efficiently parallelize MVMs. The idea of a weight stationary <sup>14</sup> dataflow is adopted to divide the workload of RNN computations and minimize the data movement of weights given the RNN workload tends to be memory-bound. Therefore, each PE will initially store fractions of the weight matrix in their respective weight buffer, and during computation, the global buffer and PEs exchange input and output activations. The second case is handled by the global buffer (GB) unit, which houses the input and output activations and contains several specialized functional units to handle across-time-step seq2seq computations such as normalization, pooling and attention.

A vector size of 16 is applied to every part of the FlexASR design. That is to say, the operations in the PE or GB are always effected on 16-element vectors or involve multiplication between a 16x16 matrix and a 16-element vector. Larger vector sizes produce higher accelerator throughput at the expense of reduced granularity for the RNN hidden state size. Therefore, the size of the RNN hidden state programmed in FlexASR is constrained to be a factor of 16 – although one may zero-pad a

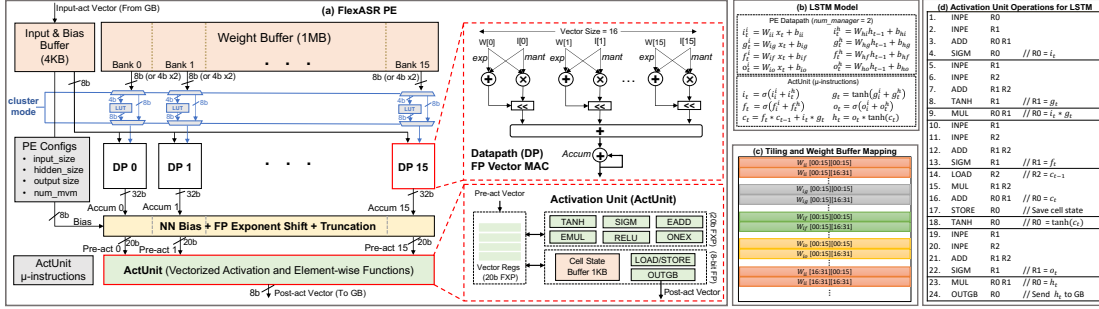


Figure 4.5: (a) FlexASR processing element (PE) highlighting its FP vector MAC and activation unit (ActUnit). For the (b) LSTM example, we adopt (c) a custom interleaved tensor tiling in the weight buffer for (d) hazard-free vector operations in the ActUnit.

non-compliant tensor shape in software prior to acceleration with FlexASR. Our choice of tile size is also influenced by the design of memory instances and the 128-bit AXI format. For example, a tile of input/weight has  $16 * 8\text{-bit} = 128\text{-bits}$ , which motivates a memory bank design with a data width of 128-bit per entry such that an AXI operation can access the full row in this memory bank.

#### 4.2.1 PROCESSING ELEMENT

The PE (Fig. 4.5(a)) is the computational workhorse of FlexASR during RNN, LSTM, GRU or linear layer computations. It contains a 1MB 16-banks weight buffer and a 4KB input and bias buffer for storing the MVM operands in 8-bit floating-point (FP8) precision. FlexASR FP8 format is E4M3 (i.e. 1-bit sign, 4-bit exponent, and 3-bit mantissa) without support for denormals. This FP8 format yielded the best accuracy outcomes after performing a search on the optimal exponent bitwidth to satisfy the dynamic range requirements of LAS models. The PE also provides alternative support for weight clustering implemented using look-up tables (LUTs) mapping 4-bit weight indexes to their 8-bit centers. This enables  $2 \times$  storage compression in the PE weight buffer.

Each weight buffer bank has a read port that feeds into a floating-point vector MAC that provides scalability along a vector dimension of 16 (similar to the PE architecture in <sup>172</sup>).

Therefore, each PE instantiates 16 vector MACs in total (i.e.,  $256$  MACs/cycles), to perform

**Table 4.2:** Vector operations supported in the FlexASR Activation Unit

Command	Description
ADD A <sub>2</sub> A <sub>1</sub>	$A_2 = A_2 + A_1$
MUL A <sub>2</sub> A <sub>1</sub>	$A_2 = A_2 * A_1$
SIGM A <sub>2</sub>	$A_2 = \text{Sigmoid}(A_2)$
TANH A <sub>2</sub>	$A_2 = \text{Tanh}(A_2)$
RELU A <sub>2</sub>	$A_2 = \text{ReLU}(A_2)$
ONEX A <sub>2</sub>	$A_2 = 1 - A_2$
COPY A <sub>2</sub> A <sub>1</sub>	$A_2 = A_1$
INPE A <sub>2</sub>	Get accumulation result from PECore into A <sub>2</sub>
STORE A <sub>2</sub>	Store cell state from A <sub>2</sub> into ActUnit Buffer
LOAD A <sub>2</sub>	Load cell state from ActUnit Buffer into register A <sub>2</sub>
OUTGB A <sub>2</sub>	Send A <sub>2</sub> to Global Buffer

MVMs between a FP8 weight vector and a FP8 activation vector.

To boost the dynamic range and accuracy of quantized RNN computations, the 32-bit fixed-point accumulated sum is dynamically shifted, at a per-layer granularity, by the AdaptiveFloat exponential bias, *expbias*. As described in Chapter 3, the latter is extracted from the maximum absolute value in the layer’s weight matrix and then stored in the FlexASAR PE registers. This allows resilient and near-FP<sub>32</sub> accuracy at FP8 precision on seq2seq models exhibiting wide parameter distribution<sup>134</sup>. After layer-wise adaptive floating-point exponent shift, the partial sums are then post-processed by the PE activation unit (ActUnit).

The ActUnit performs a sequence of vector operations (Table 4.2) to compute the necessary activation functions (e.g., sigmoid, tanh, ReLU) and the element-wise addition and multiplication of matrix-vector products coming out of the truncation unit. Fig. 4.5(c) shows the tiling convention in the multi-bank weight buffer and the ensuing sequence of ActUnit commands required to fully compute LSTM kernels without encountering logical hazards.



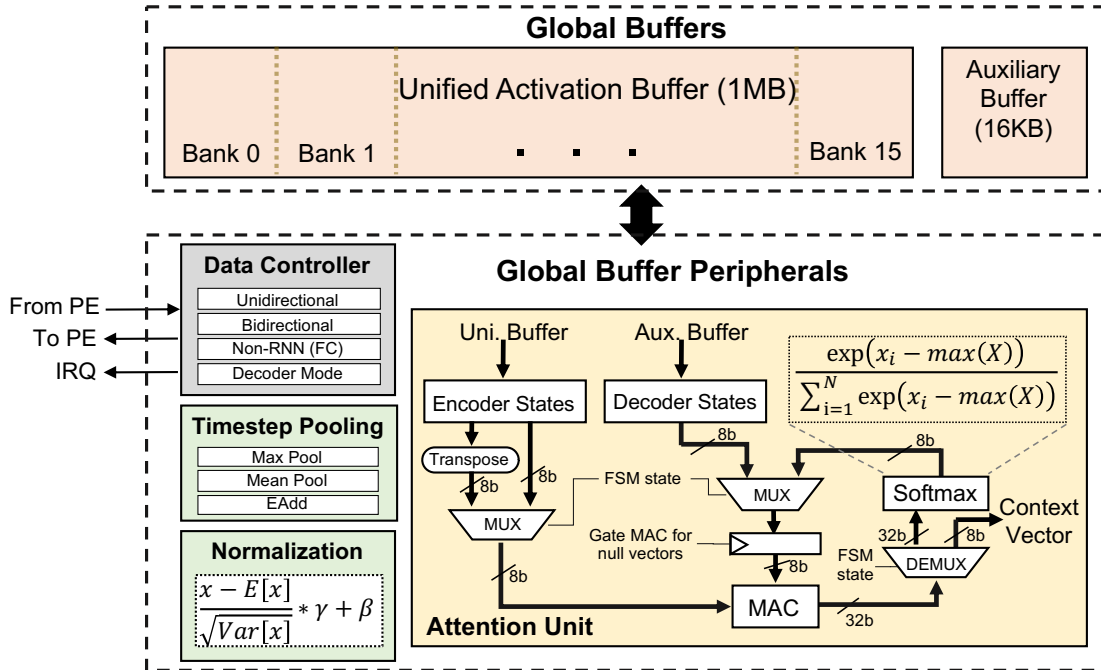


Figure 4.6: Macro-architecture of the FlexASR multi-function GB unit.

#### 4.2.2 MULTI-FUNCTION GLOBAL BUFFER UNIT

The FlexASR global buffer (GB) unit collects and unifies, across time steps, the partial RNN output states that the PEs compute. Once the partial RNN outputs for a time step are fully aggregated, the GB then broadcasts the complete activation back to each PE for the next time step computation. Moreover, the GB is augmented with auxiliary processing units that compute the attention mechanism, mean and max pooling, and layer normalization, all of which are commonly invoked in modern seq2seq NLP networks. We note that the normalization and the *softmax* operations used during the attention calculation contain several serial operations (e.g., running average), therefore DNN accelerators often offload these computations to a nearby CPU due to the lack of parallelism opportunities. We propose to compute them within the confines of FlexASR in order to reduce CPU-accelerator inter-layer activation exchanges, and thereby avoiding undue latencies during the end-to-end inference of the seq2seq model.

Fig. 4.6 shows the macro-architecture of the FlexASR GB. A 1MB 16-banks unified buffer is used to store the partial RNN hidden states computed by the PEs across time steps. This capacity is large enough to fully store thousands of activation time steps at any given time, corresponding to more than 200 words of speech, and allowing inference of large vocabulary applications requiring nuance and context understanding, especially in long sequence transductions.

While the unified buffer has a single write port, each bank has its own independent read port. A 16KB auxiliary buffer is used to house the attention intermediate states and the learnable normalization parameters. Load/store accesses to the 1MB and 16KB buffers are controlled by a GB manager which responds to requests from the PEs and the auxiliary processing modules. The latter is composed of:

- **The RNN Control Unit** which orchestrates the sequencing of the configured RNN flow mode (i.e., uni-directional, bidirectional, and seq2seq decoder mode) between the PEs and GB units. For this purpose, the RNN control module uses the configured number of time steps and the RNN hidden state size to track its job progress.
- **The Attention Mechanism Unit** which computes the soft attention mechanism<sup>8</sup> for each decoding time step. During this phase, encoder and decoder states are read from the GB unified and auxiliary buffers, respectively, before MAC operations generate scores processed by a *SoftMax* unit to produce the attention weights. To prevent numerical instability, the *SoftMax* is computed by subtracting the max score in the numerator and denominator. The attention context vector is then obtained by multiplying the attention weights with the transposed encoder states. Algorithm 2 details the step-by-step vectorized computation of the attention unit.
- **The Layer Reduction Unit** which can be configured to perform mean or maximum pooling on the RNN activations, as well as element-wise addition of the forward and backward

time steps during the bidirectional mode. Notably, *Concat*, *sum*, and *average* merge modes used during bidirectional RNNs, are supported by striping forward and backward time-steps across alternate banks in the GB unified activation buffer. For the sum or average merge modes, the GB layer reduction module performs element-wise addition (EADD) or averaging on concatenated activations.

- **The Normalization Unit** which computes layer normalization<sup>7</sup> on the RNN activations in order to speed up the training process. During the seq2seq inference, an hidden state activation is normalized as:

$$X_{norm} = \frac{X - E[X]}{\sqrt{\text{Var}[X]}} * \gamma + \beta \quad (4.1)$$

where  $\gamma$  and  $\beta$  are learnable parameters obtained after training and stored in the GB auxiliary buffer. The normalization module first computes the mean,  $E[X]$ , by running average over the number of hidden states, then evaluates the variance,  $\text{Var}[X]$ , as:  $E[X^2] - E[X]^2$ . This process gets repeated for all the needed time steps.

Finally, we note here that the attention, pooling, and normalization datapaths vectorize their computations with a vector size of 16 in order to accelerate sequential operations.

### 4.3 POST-SILICON MEASUREMENT RESULTS

An annotated photograph of the 25-mm<sup>2</sup> SM6 die is shown in Fig. 4.7 (a). The SM6 die was implemented in TSMC 16-nm FinFET technology, and flip-chip bonded into a 672-pin BGA package (Fig. 4.7 (b)). In order to orchestrate various energy efficiency ranges, six clock domains capable of outputting fine-grained frequencies, and five power domains with a 0.55-1.0V functional operation range – are servicing the main compute clusters and other on-chip endpoints. An on-chip DCO embedded in the Mo subsystem generates clocks with fine-grained frequencies to each clock

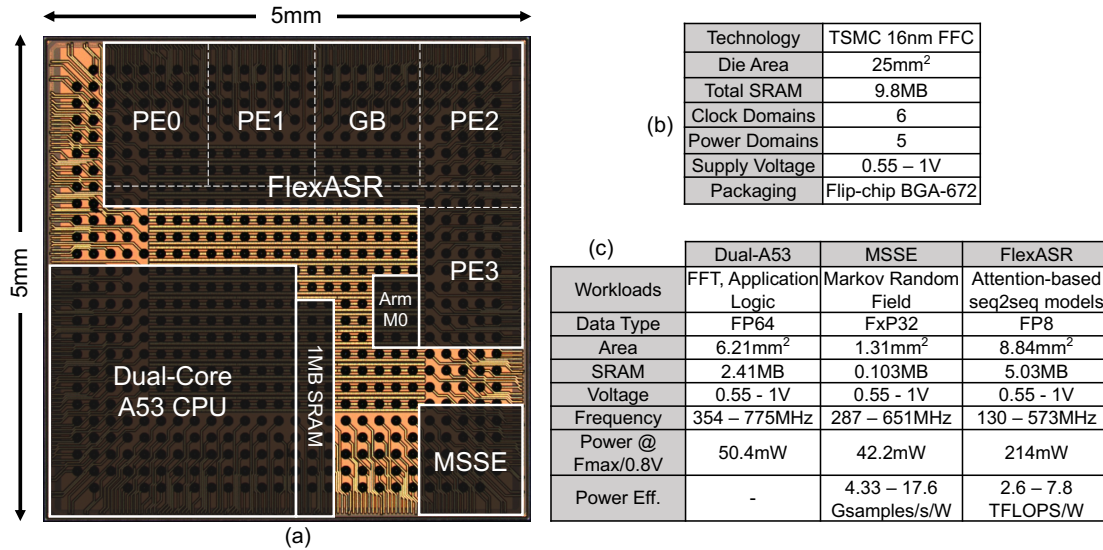
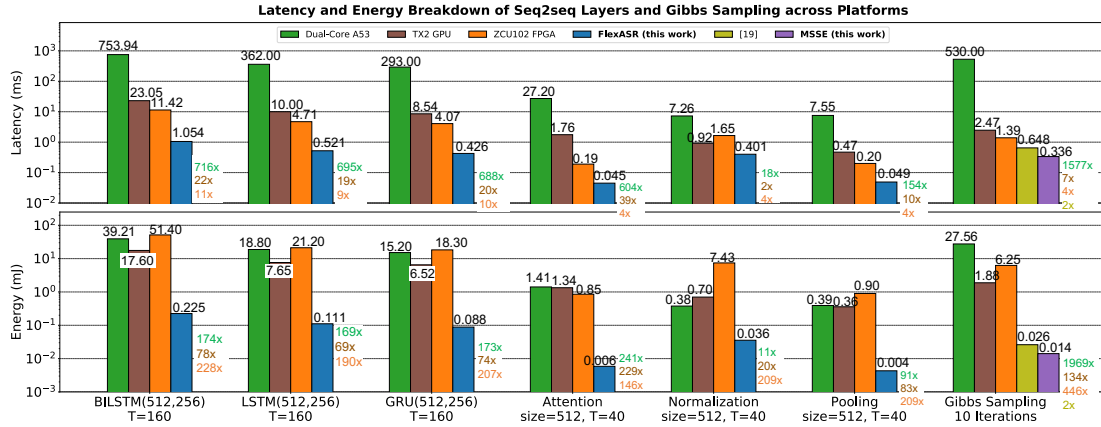


Figure 4.7: (a) Annotated die photograph of the 25-mm<sup>2</sup> SM6 test chip, and summary of (b) the SoC, and (c) its main compute clusters.

domain. A 5-V DC supply connects to the test chip PCB which then provides regulated supply voltages to each power domain. The test chip PCB attaches to a Xilinx KCU105 board using the FPGA mezzanine card (FMC) connectors. The role of the FPGA is to provide access to DRAM memory where workload data are initially stored. A UART-to-USB signaling interface allows a host laptop to communicate with the test chip.

Fig. 4.7 (c) lists the technical specifications of the main compute clusters. Notably, at 0.8-V nominal voltage, the A53 cores, MSSE, and FlexASR dissipate 50.4 mW, 42.2 mW, and 214 mW, respectively, at their maximum operating frequencies. With the compute clusters inactive, SM6 has a standby power of  $\sim 4$  mW, as the M0 remains active to sense the GPIO pins for audio detection.

To compare SM6 against commodity edge platforms, we evaluated speech-to-text LAS models and Gibbs sampling on an Nvidia TX2 mobile GPU, a Xilinx ZCU102 FPGA, and the integrated dual-core A53 CPU. TX2 results were obtained from CUDA implementations on the GPU module in order to reap the benefits of parallelization. The ASIC RTL of FlexASR and MSSE were programmed on the ZCU102 platform for evaluating FPGA performance. The Ne10<sup>142</sup> and eigen<sup>141</sup>



**Figure 4.8:** Breakdown of latency (top row) and energy (bottom row) for individual seq2seq layers running on FlexASR and Gibbs sampling running on MSSE, compared to running on different commercial platforms. Here, FlexASR, MSSE, and the A53 cores are running at frequencies of 440MHz, 533MHz, and 715MHz, respectively, at 0.8V.

libraries were used to vectorize supporting ASR and Gibbs sampling kernels on the A53 SIMD units. We conducted the following application-level measurements at room temperature using a typical silicon.

#### 4.3.1 PER-LAYER CHARACTERIZATION

We begin by characterizing the processing times and energy dissipation of the main SM6 compute clusters while running individual seq2seq layers and Gibbs sampling iterations (Fig. 4.8). We can make the following observations:

(1) FlexASR provides significant speedup gains while accelerating seq2seq layers – with attention, LSTM and GRU RNNs showing greater benefits. Notably, the 160-time steps bidirectional LSTM (BILSTM) exhibits higher processing speedup over CPU, GPU, and FPGA compared to the unidirectional LSTM scenario – due to FlexASR striping forward and backward activations in alternate banks in its global buffer unit. In addition, even though the normalization and pooling operations are very serial in nature, by specializing their datapaths within the confines of the accelerator and thereby avoiding accelerator-CPU activation exchanges – FlexASR still outperforms all

other platforms.

(2) MSSE achieves appreciable latency reductions over the commercial edge platforms – demonstrating the need for specialized Gibbs sampling accelerated computing as the A53 cores, TX2 GPU, and ZCU102 are  $1577\times$ ,  $7\times$ ,  $4\times$  slower than MSSE, respectively. Moreover, as MSSE was optimized for Bayesian inference with binary labels as opposed to the general purpose PGMA accelerator<sup>71,72</sup> (supporting up to 64 labels), Gibbs sampling runs twice as fast on MSSE.

(3) Both FlexASR and MSSE generates orders-of-magnitude smaller energy consumption compared to the commercial edge devices. This is particularly striking during Gibbs sampling as the A53 cores, TX2 GPU, and ZCU102 produce  $1969\times$ ,  $134\times$ , and  $446\times$  larger energy dissipation, respectively, compared to MSSE. Furthermore, although the FPGA generally executes seq2seq kernels faster than the dual-core A53 and TX2 GPU, its power consumption envelope is significant enough to make it the least energy-efficient platform for several workloads (e.g., BILSTM, GRU, pooling).

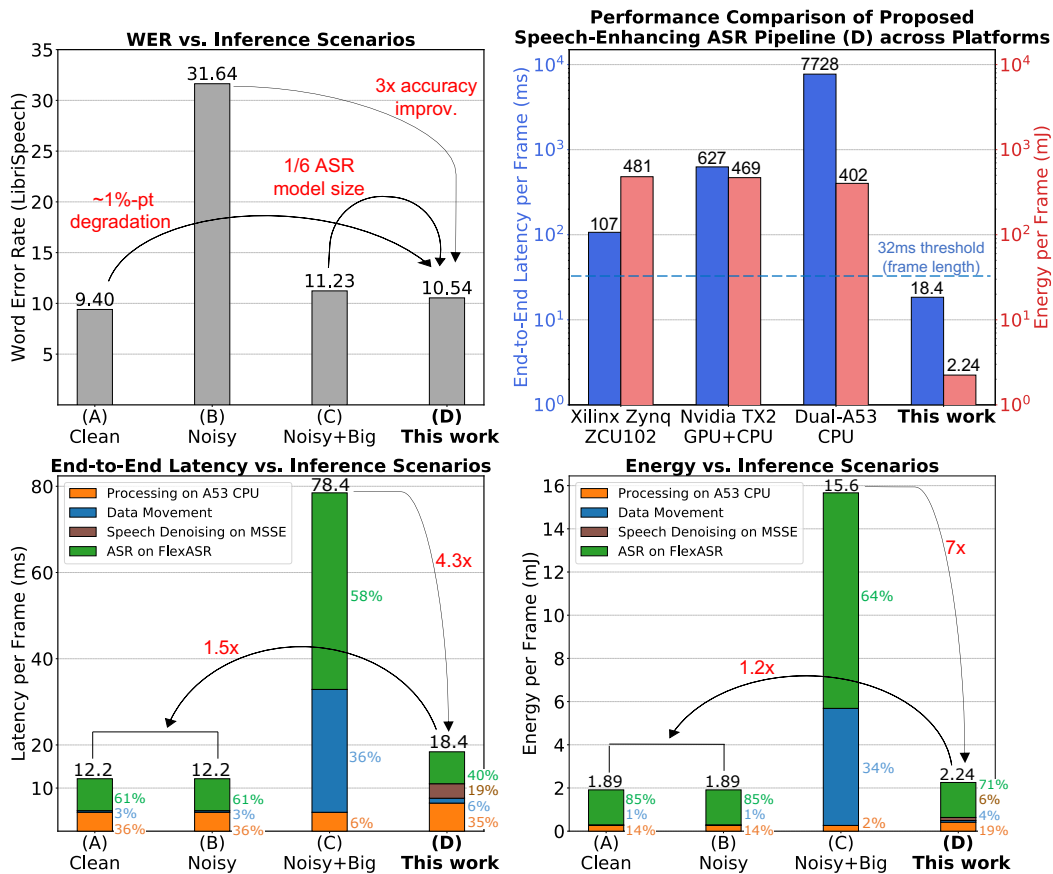
Finally, we note that the RNN (BILSTM, LSTM, or GRU) and linear workloads, which are computed in the FlexASR PEs, achieve near 100% MAC utilization. However, at the overall end-to-end ASR workload level, PE utilization is about 71% given the PEs become idle during the computation of normalization, pooling, and attention which account for 19%, 6%, and 4%, respectively, of a representative ASR workload<sup>†</sup>.

#### 4.3.2 END-TO-END CHARACTERIZATION

To demonstrate the accuracy and end-to-end performance benefits of the proposed speech-enhancing pipeline, we compare our approach (Scenario D) with three other common inference scenarios using the LibriSpeech dataset<sup>100</sup> as shown in Fig. 4.9. Scenario A emulates a clean environment wherein the speaker’s voice is the single audio source. Scenario B mixes the speaker’s voice with another human voice source in a simulated room environment for a signal-to-noise ratio (SNR) of

---

<sup>†</sup>ASR model used in Scenario A, B, and D from Fig. 4.9.



**Figure 4.9:** End-to-end measurement results for ASR inference with (A) clean input audio, (B) noisy input audio, (C) noisy input audio using 6x larger ASR model, and (D) this work - noisy input audio with Bayesian sound source separation denoising. Word error rate (WER) performance (top left), end-to-end per-frame latency (bottom left), energy (bottom right), and cross-platform comparisons (top right) are shown.

0.90 dB. Finally, Scenario C (*Noisy+Big*) adopts a much larger ASR model (22 MB vs. 3.5 MB used in Scenario A, B, and D) trained with a noise-corrupted LibriSpeech dataset in order to learn from noisy inputs. The *Noisy+Big* model was gradually sized up, by increasing the hidden state dimension, until its WER is much closer to scenario A in noisy cases. The ASR LAS model adopted in Scenario A, B, and D consists of 4 four BILSTM RNN stacks in the encoder with 512 cells and a 256-cell unidirectional LSTM RNN in the decoder. The LAS model in Scenario D was scaled up to 1024 cells in each of the four BILSTM RNN stack in the encoder and 1024 cells in its decoder

LSTM RNN unit. The following observations are made:

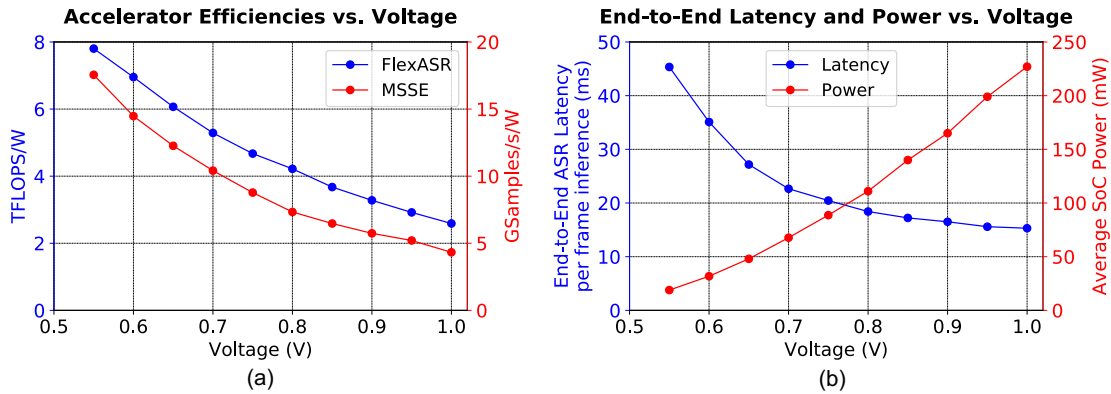
(1) By denoising incoming audio signals and preceding the speech-to-text inference, MSSE allows FlexASR to compute significantly smaller-size (up to  $6\times$  smaller), iso-accurate ASR models trained on widely-available single-source clean datasets (Fig. 4.9 top left) – obviating the very inefficient strategy of scaling up the DNN model size (Scenario C) in order to achieve noise robustness. Furthermore, the proposed ASR pipeline delivers  $3\times$  accuracy improvement over the unseparated noise case (Scenario B) and is within 1% of the clean input baseline case (Scenario A). We note that in Scenario D, MSSE executes 150 Gibbs sampling iterations, improving speech quality by up to 7.3dB SDR.

(2) The proposed pipeline achieves  $4.3\times$  lower end-to-end per-frame latency (bottom left), and  $7\times$  energy improvement (bottom right) compared to the similarly-accurate case in Scenario C, which requires significant off-chip data movements because the weights of the upsized ASR model cannot fully fit in FlexASR PE scratchpads. Notably, SM6 achieves a latency per frame of 18.4-ms while dissipating 2.24-mJ of energy during the end-to-end speech-enhancing ASR inference.

(3) Due to the use of RNNs, the inference computation is mainly memory-bound. Therefore, it can be observed that the latency and energy costs of memory transfers in the bigger ASR model (i.e., Noisy+Big in Scenario C) are much higher compared to the leaner ASR model used in Scenario A, B, and D. For example, memory transfers in Scenario C account for 36% of the end-to-end latency vs. only 6% in our proposed pipeline whose RNN model is  $6\times$  smaller.

(4) Fig. 4.9 (top right) shows that despite substantial energy expenditures, the commercial edge platforms fail to provide real-time performance as their per-frame latencies exceed the 32ms frame length.





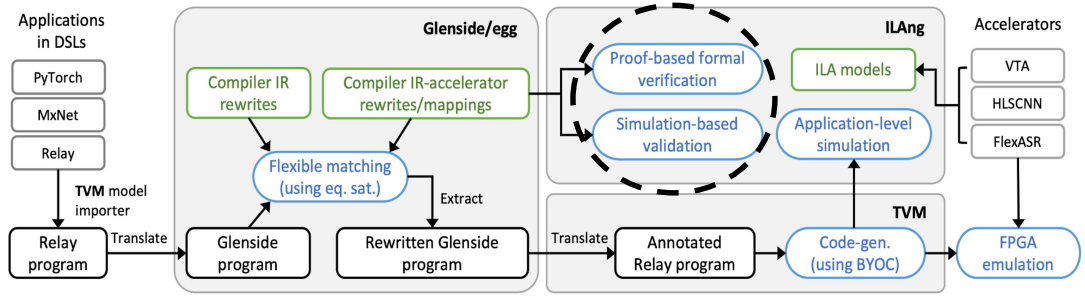
**Figure 4.10:** Impact of voltage scaling on (a) accelerators' power efficiency, and (b) end-to-end ASR latency and SoC power envelope.

### 4.3.3 VOLTAGE SCALING

To evaluate the functional efficiency range of the SoC, all the power domains are uniformly scaled from 1.0-V down to 0.55-V while the various compute clusters (FlexASR, MSSE, dual-A53) are clocked at their respective maximum frequencies. Fig. 4.10 (a) shows that voltage/frequency scaling on FlexASR and MSSE produces efficiency ranges of 2.6-7.8 TFLOPs/W and 4.33-17.6 GSamples/s/W, respectively. The per-frame, end-to-end latency varies from 45ms to 15ms as the SoC voltage scales from 0.55-1.0V, while consuming 19-227mW on average (Fig. 4.10 (b)). At nominal 0.8-V, the average per-frame SoC power is 111 mW.

### 4.4 VERIFYING FLEXASR VIA A FORMAL HARDWARE/SOFTWARE COMPILER INTERFACE

In order to fully exploit the benefits of hardware specialization, we must seamlessly lower the high-level application code to these accelerators. And doing that, unfortunately, is not as easy as it sounds. First, we must identify the various functions of the application code that are supported by these accelerators and then ensure computation integrity after lowering the compilation code down to the hardware device.



**Figure 4.11:** Prototype implementation of the D2A compilation flow, highlighting the proof-based and simulation-based verification components.

While programming FlexASR can be easily done via AXI commands over memory-mapped I/Os, compiling down to it, however, can be challenging because as most deep learning domain specific languages (DSLs) are concerned with individual tensor operations, a single FlexASR instruction may, in fact, correspond a full neural network layer (e.g. LSTM, GRU, LayerNorm). So, this requires the compiler to deal with the dramatic granularity mismatch between the high-level application code and the low-level hardware details. Furthermore, what makes FlexASR compilation an even trickier challenge, is that, computations are performed according to a custom numerical data type: `AdaptivFloat`. So, all of the mechanics surrounding this custom number system, have to be embedded somewhere inside the compiler.

The current common practice is to design a dedicated driver for each accelerator and a specialized compiler stack for each compiler framework. This requires significant amount of human labor because engineering effort are needed for every device on every compiler framework.

Together with researchers from Princeton and the University of Washington, we propose to address these mapping challenges with a compiler flow called D2A<sup>54</sup> which uses Instruction-Level Abstraction (ILA)<sup>55</sup> as a formal software/hardware interface. As a demonstration of the D2A methodology, we have implemented an end-to-end compiler pipeline for deep learning applications, as shown in Figure 4.11, in order to enable flexible, portable, and verifiable compiler support for

	Accelerator	Operation	Avg. Err.	Std. Dev.
1	VTA	GEMM	0.00%	0.00%
2	HLSCNN	Conv2D	1.78%	0.16%
3	FlexASR	LinearLayer	0.84%	0.29%
4	FlexASR	LSTM	1.21%	0.19%
5	FlexASR	LayerNorm	0.27%	0.20%
6	FlexASR	MaxPool	0.00%	0.0%
7	FlexASR	MeanPool	1.79%	0.28%
8	FlexASR	Attention	4.22%	0.09%

**Figure 4.12:** Simulation-based validation results of checking IR-accelerator mappings. The average relative error (Avg. Err.) and the standard deviation (Std. Dev.) of the errors are measured for simulation over 100 test inputs.

new hardware accelerators. The ILA is an ISA-like formal model. It generalizes the instruction notion from the processor world and apply it to accelerators. Utilizing the automatically generated accelerator software model from their ILA models, we are able to run fast end-to-end application-level simulation to provide feedback on application deployment, which are valuable for both application development and accelerator designs.

Figure 4.12 highlights the validation results when applying D2A to FlexASR, thereby checking the IR-accelerator mappings for these non-trivial operations. We observe very close matching between the frontend DSL and the hardware numerical outputs – with deviations mainly caused by the accumulation of the 8-bit AdaptiveFloat quantization errors which are not factored in the frontend TVM translation.

While this section described our approach to efficiently computing attention-based seq2seq RNNs, which have been applied to speech-to-text and machine translation applications, the next section lays out our algorithm-architecture-silicon co-design principle to lower the prohibitive cost of inferencing Transformer-based models – which are now the most widely used neural ingredients in modern NLP.

#### 4.5 TAKEAWAYS

In this work, we demonstrated the usefulness of optimizing speech-to-text execution at the granularity of a system whereby the underlying machine learning algorithms are carefully leveraged and tweaked to promote higher overall energy efficiency. In particular, we showed that we can obviate the need to utilize large ASR neural networks (along with the ensuing higher latency and energy) by plugging in the right algorithmic ingredient in the front-end execution pipeline of a monolithic system-on-chip – and doing so without compromising the end-to-end task accuracy. The latter is boosted by the effectiveness of the unsupervised Gibbs sampling algorithm, the higher representation power of the AdaptiveFloat number system, and the greater neural performance afforded by the attention mechanism.

---

**Algorithm 2:** Computation Steps of the Attention unit.

---

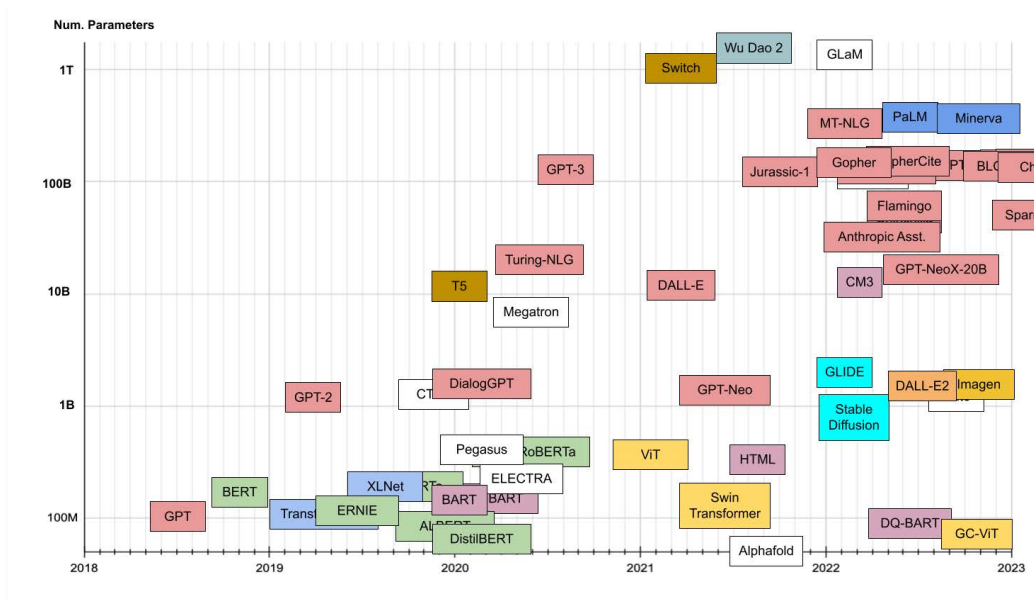
**Input:** Encoder Matrix  $M$ , Decoder Vector  $v$   
**Output:** Attention Context Vector  $A$   
 $N_T$  := encoder time steps in tiles  
 $N_D$  := decoder size in tiles  
 $max = -\inf$   
*// 1st MV Mult*  
**for**  $i = 0$  to  $N_T - 1$  **do**  
     $accum := 0$   
    **for**  $j = 0$  to  $N_D - 1$  **do**  
         $W := M_{[16i:16i+15][16j:16j+15]}$   
         $v := v_{[16j:16j+15]}$   
         $accum += W * v$   
    store  $accum$  to auxiliary buffer  
    *// Get maximum at the same time of 1st stage*  
    **if**  $max < max(accum)$  **then**  
         $max = max(accum)$   
    *// Denote output of 1st MV Mult as X*  
     $X_{[16i:16i+15]} = accum$   
*// Softmax step 1: SRAM read on X*  
     $sum_{exp} = 0$   
    **for**  $i = 0$  to  $N_T - 1$  **do**  
         $sum_{exp} += sum(exp(X_{[16i:16i+15]} - max))$   
*// Softmax step 2: SRAM read/write to get result X'*  
    **for**  $i = 0$  to  $N_T - 1$  **do**  
         $X'_{[16i:16i+15]} = (X_{[16i:16i+15]} - max) / sum_{exp}$   
*// 2nd MV Mult*  
    **for**  $i = 0$  to  $N_D - 1$  **do**  
         $accum := 0$   
        **for**  $j = 0$  to  $N_T - 1$  **do**  
             $W := M^T_{[16i:16i+15][16j:16j+15]}$   
             $v := X'_{[16j:16j+15]}$   
             $accum += W * v$   
        store  $accum$  to GB auxiliary buffer  
        *// Context vector, A, is output of 2nd MV Mult*  
         $A_{[16i:16i+15]} = accum$

---

# 5

## Lowering the Cost of Inferencing Large Language Models on Embedded Devices

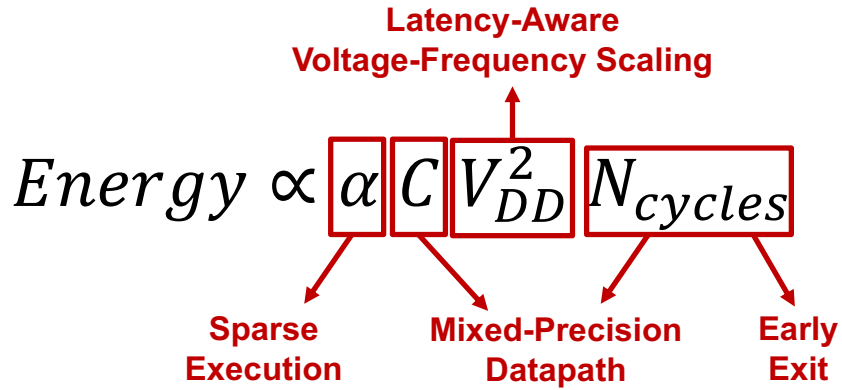
HOW TO HARNESS THE POWER OF LARGE LANGUAGE MODELS ON LOW CAPACITY DEVICES IN AN ENERGY-EFFICIENT MANNER? Large language models such as BERT are fueling the growth of intelligent virtual assistants, which leverage natural language processing (NLP) to implement inter-



**Figure 5.1:** The unabated pursuit for omniscient and omnipotent AI is currently driving large language models (LLMs) to extreme scales. Image source: Transformer models: an introduction and catalog<sup>3</sup>

active voice interfaces. At the same time, the incessant pursuit of greater linguistic representation is currently driving NLP models to extreme scales – with recent large language models (LLMs) featuring more than a trillion parameters as illustrated in Figure 5.1. Currently, these hefty NLP models are offloaded to the cloud. However, it is desirable to deploy them on edge devices, where personal data can be kept private and the round trip latency to the cloud is removed. The constraints on mobile can be quite different to the datacenter scenario. Firstly, since we are dealing with user input, we need to meet real time throughput requirements to prevent a noticeable lag to the user. Secondly, energy consumption is a critical concern on edge/mobile devices.

Our work, EdgeBERT<sup>131,135</sup>, offers a principled latency-driven approach for deploying Transformer workloads onto edge devices with minimal energy consumption thanks to early exit prediction and entropy-controlled voltage-frequency scaling (VFS). While the benefits of early exit can be reaped on commodity GPUs, we unlock additional energy savings by co-designing the hardware



**Figure 5.2:** Summary of the energy reduction strategies adopted during Transformer NLP inference.

datapaths. In addition, as shown in Figure 5.2, sparse execution is performed during the mixed-precision (FP4/FP8) computation in order to further reduce energy overheads.

These energy-lowering measures are controlled by a statistical metric called entropy, which can be understood as a *gauge* of the classification confidence. In the next section, we explain how the entropy is leveraged to eliminate undue NLP energy and latency costs.

### 5.1 ENTROPY-BASED EARLY EXIT

The motivation behind early exit (EE) is to match linguistically complex sentences with larger (or deeper) models and simple sentences with smaller (or shallower) models<sup>159</sup>. This is typically done by adding a lightweight classifier at the output of the Transformer layer so that a given input can exit inference earlier or later in the stack, depending on its structural and contextual complexity. The classifier computes and compares the entropy of an output distribution with a preset “confidence” threshold,  $E_T$ , in order to assess whether the prediction should exit or continue inference in the next Transformer encoder layer. The entropy metric quantifies the amount of uncertainty in the data. Smaller entropy values at a Transformer layer output implies greater confidence in the correctness of



the classification result. The entropy  $H$  on sample  $x$  is estimated as:

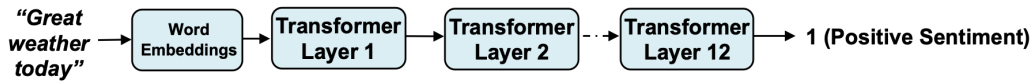
$$H(x) = - \sum p(x) \log p(x) = \ln \left( \sum_{k=1}^n e^{x_k} \right) - \frac{\sum_{k=1}^n x_k e^{x_k}}{\sum_{k=1}^n e^{x_k}} \quad (5.1)$$

The early exit condition is met when  $H(x) < E_T$ . Therefore, the larger  $E_T$  becomes, the earlier the sample will exit (i.e.  $N_{cycles}$  becomes smaller) with potentially lower accuracy.

For this analysis, we consider the state-of-the-art *BERT-base* model which is comprised of twelve serial computationally intensive transformer layers, each layer containing twelve parallel attention heads whose outputs concatenate to drive a large feed-forward network<sup>27</sup>.

Let's say we are executing a sentiment analysis task using BERT. As shown in Figure 5.3 (a), during the conventional BERT inference, the computation goes through all 12 Transformer layers before classifying the output as either a positive or a negative sentiment. However, for an input query

**(a) Conventional BERT Inference**



**(b) BERT Inference with Entropy-based Early Exit**

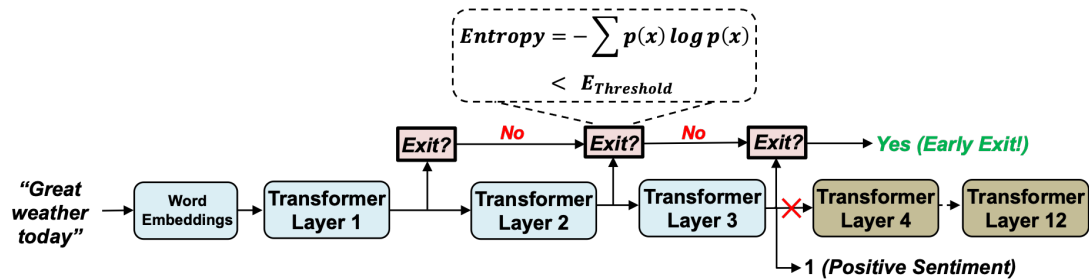
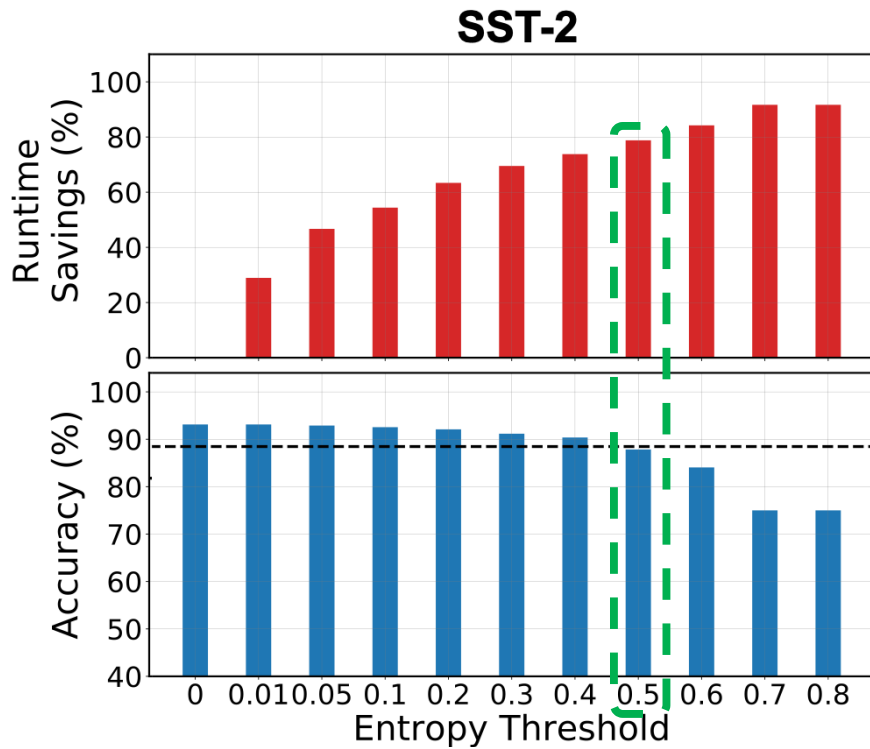


Figure 5.3: Sentiment analysis execution with the BERT Transformer model during (a) the conventional and (b) entropy-based early exit inferences.



**Figure 5.4:** By setting an entropy threshold of 0.5, 80% of Transformer computations are saved while maintaining 95% of the original BERT accuracy on the SST-2 sentiment analysis benchmark.

as simple as "Great weather today", we can leverage the entropy function to terminate the inference early, at the output of Transformer layer 3 instead of layer 12 as shown in Figure 5.3(b). Looking across the entire sentiment analysis dataset (SST-2), we observe significant opportunities for computation savings depending on the selected entropy threshold as shown in Figure 5.4. As tolerance to the model accuracy loss is very much subjective, the entropy threshold can be used as a knob by users when they consider this tradeoff space between accuracy and runtime savings.

We note that the application of the entropy-based early exit concept is not new in the case of encoder-based sequence-to-sequence Transformers given it was originally applied to CNNs<sup>138</sup>. A novelty in this work is the utilization of the entropy function for energy minimization via entropy-controlled voltage and frequency scaling, which is discussed in greater in the next section.

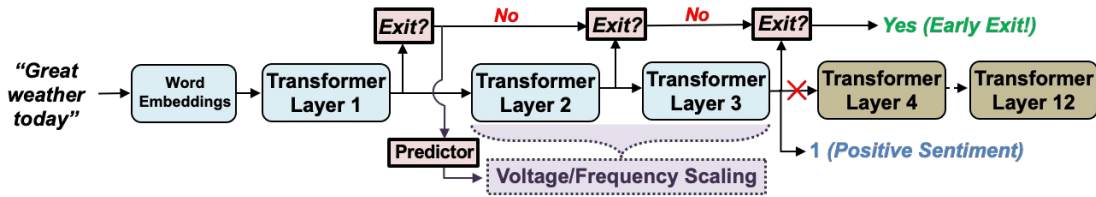


Figure 5.5: Given an end-to-end latency target, we utilize the entropy value measured at the output of the 1st Transformer layer to predict the number of additional Transformer layers the accelerator needs to compute. And, then based on this information, the EdgeBERT accelerator opportunistically scales its supply voltage and clock frequency.

## 5.2 ENTROPY-CONTROLLED VOLTAGE-FREQUENCY SCALING

Dynamic voltage frequency scaling (DVFS) is a widely used technique to dynamically scale down the voltage and frequency for less computationally intensive workloads. In the past, DVFS has been widely deployed in commercial CPUs<sup>143</sup>, and GPUs<sup>90</sup>. However, these schemes typically adjust the voltage and frequency at a coarse granularity at workload-level. In contrast, in this work, we perform **fine-grained sentence-level voltage frequency scaling** to reduce the energy consumption for NLP inference while adhering to a prescribed latency target.

The inference of a sentence starts at nominal voltage and maximum frequency, and the entropy value is calculated at the output of the first Transformer encoder layer. The entropy result is then sent to a trained multi-layer perceptron classifier (early exit predictor) to predict which following Transformer layer will mark the termination of the inference (e.g. early exit at layer 6). Based on the predicted early exit layer, the voltage and frequency is scaled down to proper energy-optimal setting for the rest of Transformer encoder layers (e.g. layer 2 to 6) **while meeting the latency target for each sentence**. This scheme, which is illustrated in Fig. 5.5, produces a quadratic reduction in the accelerator power consumption.

In the next section, we describe the EdgeBERT hardware accelerator system, which is a sparse Transformer processor specialized to efficiently compute BERT-based Transformer networks along with the entropy-based optimizations we have just discussed.

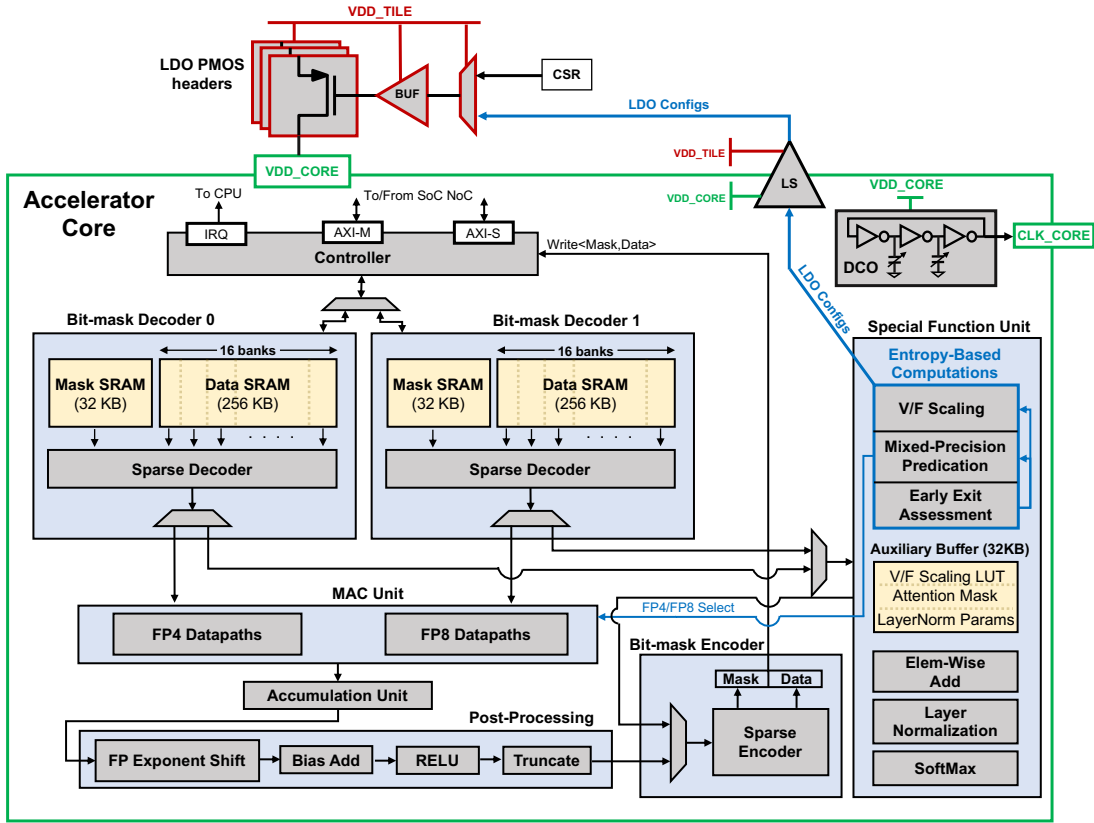
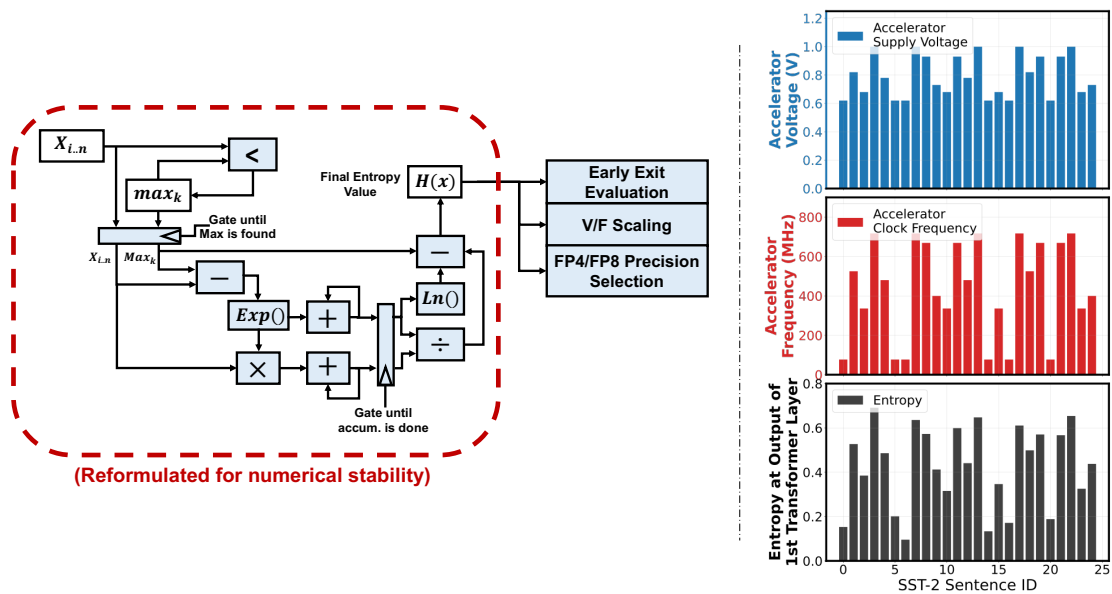


Figure 5.6: Architecture of the EdgeBERT Sparse Transformer Processor (STP).

### 5.3 THE EDGEBERT SPARSE TRANSFORMER PROCESSOR

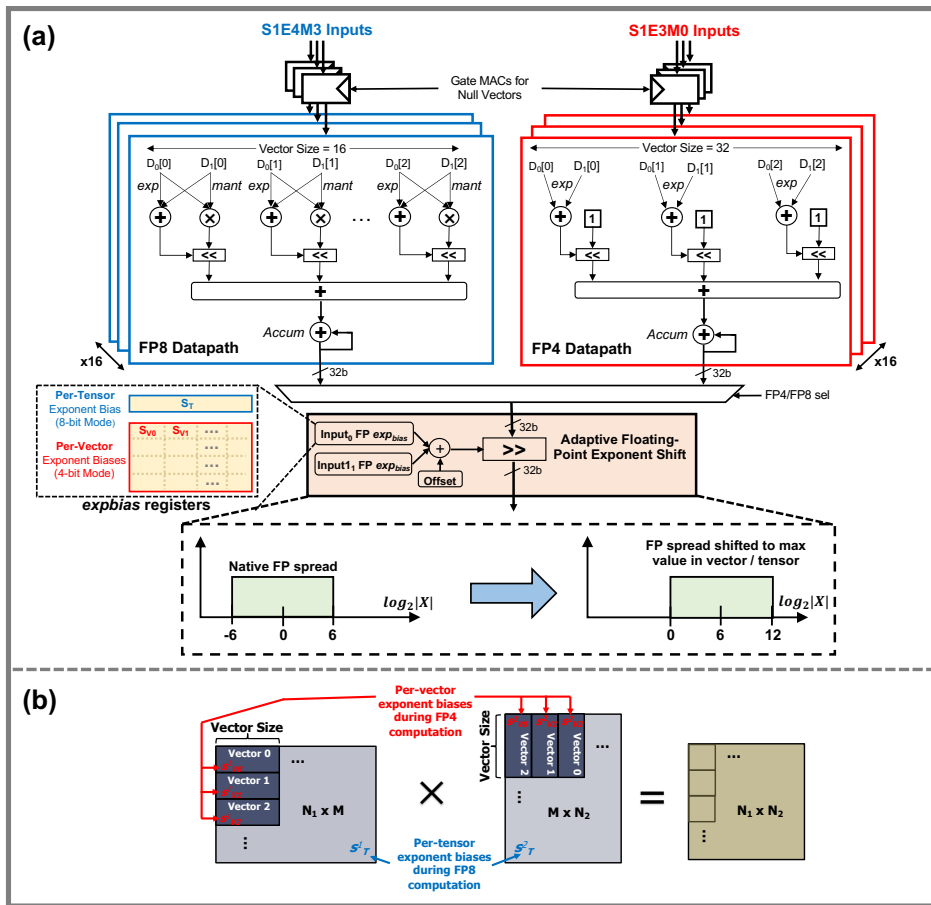
Figure 5.6 lays out the architecture of the EdgeBERT Sparse Transformer Processor (STP). The main computation engine comprises a mixed-precision MAC unit that selects between FP4 (E<sub>3</sub>M<sub>0</sub>) vs. FP8 (E<sub>4</sub>M<sub>3</sub>) execution based on the entropy characteristics of the input sentence. The entropy information further guides the power management scheme, which dynamically adjusts the accelerator’s local supply voltage via cell-based PMOS power headers of a free running LDO (i.e. feed-forward LDO with no feedback loop). Sixteen pre-characterized look-up table (LUT) entries, stored in a 32KB auxiliary buffer of the special function unit (SFU), control the effective resistance of the



**Figure 5.7:** Hardware implementation of the entropy function whose value on the 1st layer Transformer output is used to scale the accelerator’s supply voltage and clock frequency.

PMOS power headers. A configurable digitally-controlled oscillator (DCO) runs off of the local supply voltage to ensure proper self-clocked operation. The SFU further contains specialized data-paths to provide vectorized computations for various non-linear functions (i.e., softmax, layer normalization), element-wise addition, as well as the entropy computation. The SFU’s auxiliary buffer also stores normalization parameters and attention pruning metadata which specify the span of each attention head. Notably, the STP entirely skips the computation of attention heads with null span, further cutting inference latency and energy. A pair of bit-mask decoders and a bit-mask encoder decompress and compress sparse matrices, respectively, before and after each matrix-matrix multiplication, which enables compact storage of non-zero data along with their associated binary tags in the 256KB data SRAM and 32KB mask SRAM, respectively.

Figure 5.7 shows the algorithmic and hardware implementations of entropy-based early exit computation. To avoid numerical instability during the calculation of exponents, we reformulate the



**Figure 5.8:** (a) Mixed-precision FP8/FP4 processing element. The FP8 MAC has a vector size of 16 while the FP4 MAC utilizes a vector size of 32, enabling the STP to double its throughput. (b) Post-accumulation tensor scaling is performed at a per-vector, and per-tensor granularity during FP4, and FP8 computations, respectively.

entropy algorithm by negatively scaling all elements in the early exit vector by the maximum score. Figure 5.7 also plots the per-sentence correlation between the 1st layer's computed entropy and the achieved voltage and frequency scaling via the embedded VFS mechanism. We observe, here, fine-grained, **per-sentence** power optimization based on the entropy characteristics of the input query.

### 5.3.1 MIXED-PRECISION FP4/FP8 DATAPATH

Figure 5.8 (a) shows the details of the mixed-precision (MP) floating-point MAC unit. In this work, we consider two floating-point based number systems for high dynamic range computations (Figure 5.9). FP8 with a 4-bit exponent field and a 3-bit mantissa field, and FP4/LOG4 with a 3-bit exponent field and no mantissa. In this case, we essentially have a 4-bit logarithmic data type wherein we additionally use the gamma variable to control the distance between values<sup>166</sup>. Doing tensor

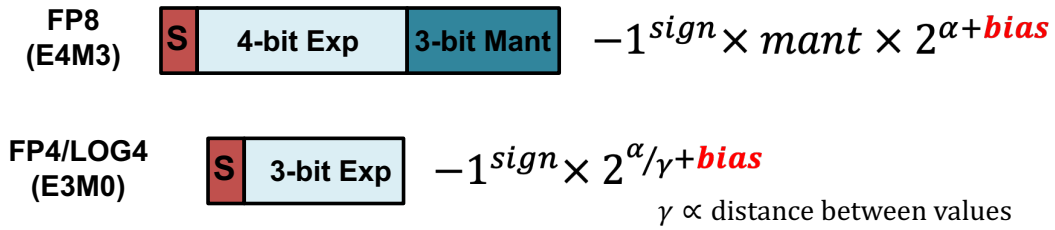


Figure 5.9: Floating-point number systems implemented in the STP mixed-precision MAC.

multiplication in a logarithmic processing element is appealing in the sense that multipliers are not needed. As illustrated in Figure 5.8 (a), the LOG4 datapath, at the core, simply needs adders and shifters, enabling greater energy efficiency.

To ensure that we can accurately compute tensors with very large dynamic ranges, we add a bias parameter in the exponent field of both data types to enable the STP to do post-accumulation tensor scaling, similar to the AdaptivFloat concept discussed in Chapter 3. However, doing a naive implementation of FP4 with per-tensor scaling leads to a steep loss in the model accuracy. To dampen this accuracy degradation in the FP4 regime, we opt to do per-vector scaling. Similar to recent work that leverages per-vector scale factors during INT4 quantization<sup>64</sup>, the STP uses per-vector exponent biases during FP4 computations (Figure 5.8 (b)).

The MP processing element comprises 16 lanes of FP8 vector datapaths with coarse-grained per-tensor exponent bias scaling, each lane using a vector size of 16. And, 16 lanes of LOG4 datapaths

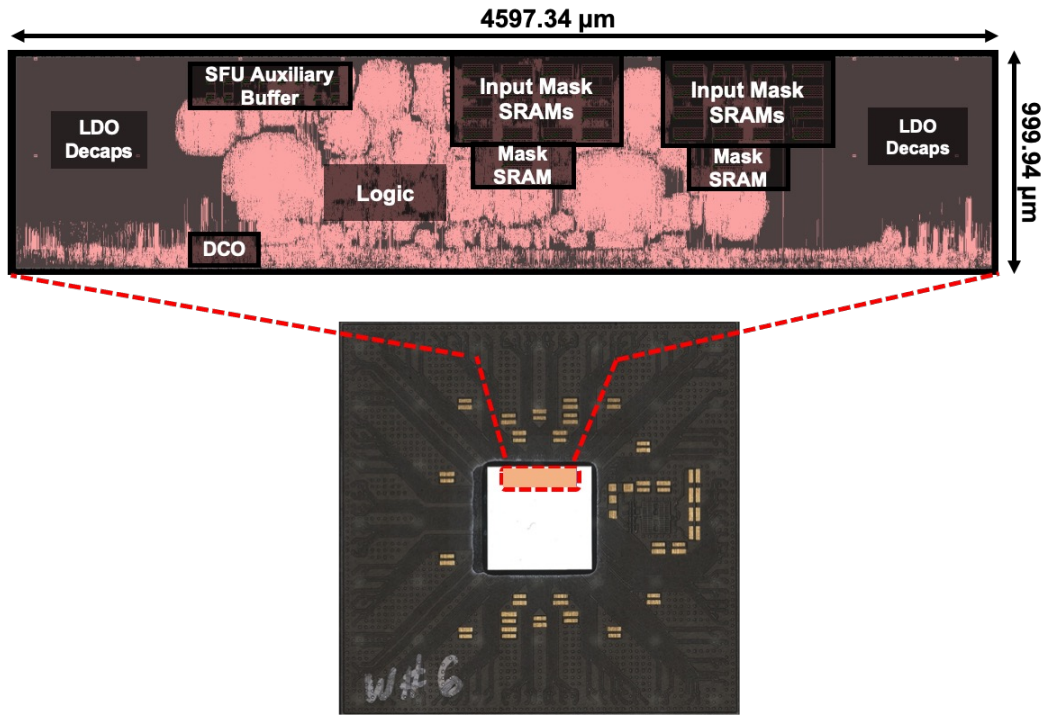


Figure 5.10: Annotated die photo highlighting the physical layout of the 12nm EdgeBERT STP.

with fine-grained per-vector exponent biases are provisioned with a vector size of 32, enabling the STP to double its throughput. The MP datapaths are gated for null input vectors in order to save power. The selection between the FP<sub>4</sub> and FP<sub>8</sub> datapaths is controlled by the SFU entropy unit. Therefore, depending on the entropy characteristics of the input query, the STP activates either higher (FP<sub>8</sub>) or lower (FP<sub>4</sub>) precision computations.

#### 5.4 12NM CHIP PROTOTYPE AND POST-SILICON MEASUREMENT RESULTS

The proposed STP was fabricated using GlobalFoundries 12nm FinFET process as part of a larger system. Figure 5.10 shows an annotated physical layout of the STP overlaid on a photograph of the flip-chip packaged die. Decaps were carefully placed at the output of the feed-forward LDO to ensure proper RC time constant compliance and adequate noise filtering.



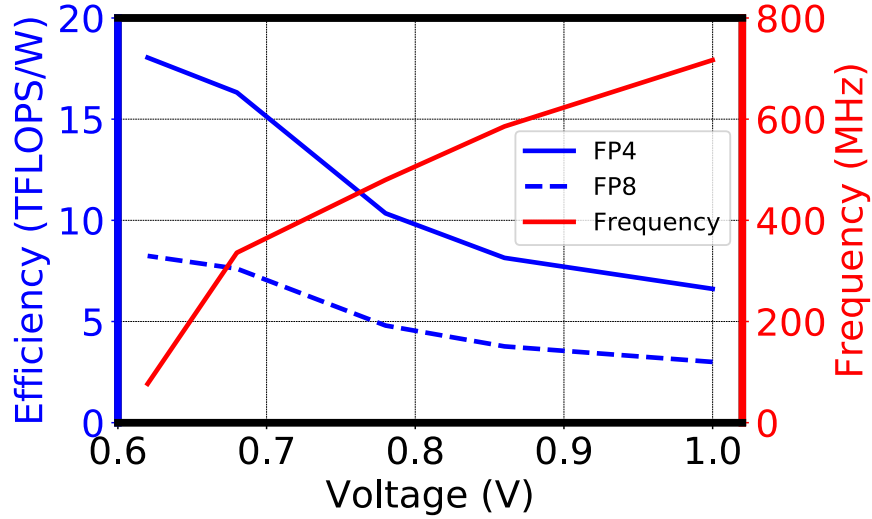


Figure 5.11: V/F scaling performance of the STP.

Figure 5.11 plots the frequency vs. voltage operation of the 4.6mm<sup>2</sup> STP and the resulting computational efficiency presented in terms of TFLOPS/W. Functional operation is verified across a 0.62-to-1.0V supply voltage range with 77-to-717MHz clock frequency, while producing an energy efficiency range of 3.0-8.24TFLOPS/W (FP8) and 6.61-18.1TFLOPS/W (FP4).

Figure 5.12 presents the post-silicon latency and energy results while executing the computations of *BERT-base* on the sentiment analysis (SST-2) dataset. Co-designing the early exit (EE) algorithm allows the accelerator to reduce latency and energy expenditures by 3.71×. Attention head pruning (AP) further extends these savings by 1.5×. The addition of mixed-precision (MP) predication enables the processor to achieve a minimum average inference latency of 682ms. Assuming an end-to-end per-sentence latency constraint of 2 seconds, the STP can opportunistically derate its voltage and frequency (VFS) to further reduce energy consumption, measured down to an average of 65mJ per inference. The horizontal histograms in Figure 5.12 provide further details of the measured per-sentence processing latency and energy results. Combining EE, AP, and MP allows the majority of sentences to execute under 1 second, while VFS allows the accelerator to process most sentences un-

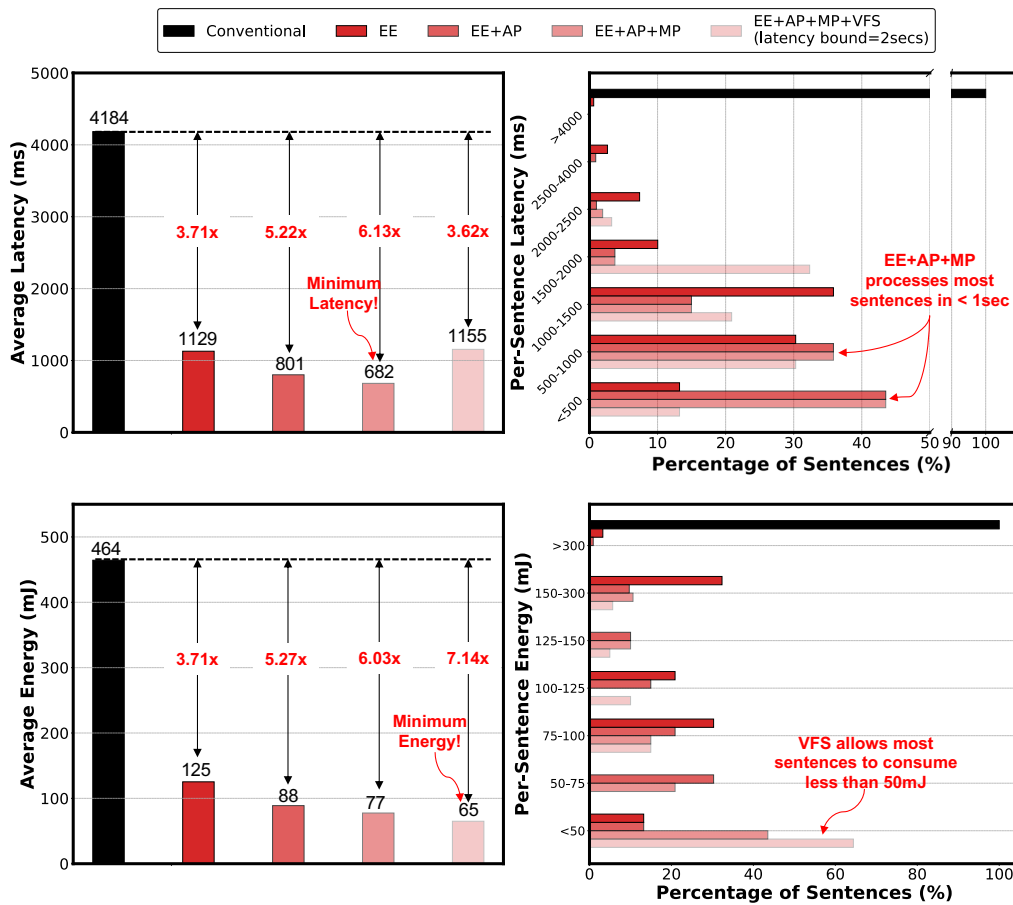


Figure 5.12: Measurement results for BERT inference on the SST-2 dataset, highlighting the benefits of early exit (EE), attention head pruning (AP), mixed-precision (MP), and latency-bounded V/F scaling (VFS).

der 50mJ/inf by relaxing the latency target to 2 seconds. In contrast, a conventional implementation would consume 4 seconds of latency and 464mJ per inference.

Finally, we note that given the modularity of the STP hardware architecture, the latter can be scaled to achieve stricter latency targets (e.g., < 100ms) by using a higher count of MAC vector sizes and parallel lanes.

## 5.5 TAKEAWAYS AND FOLLOW-ON RESEARCH

As newer Transformer-based pre-trained models continue to generate impressive breakthroughs in language modeling, they characteristically exhibit complexities that levy hefty latency, memory, and energy taxes all computing scales. The EdgeBERT algorithm-hardware co-design framework takes a latency-driven approach to minimize Transformers' energy consumption. Specifically, we adopt first-layer early exit prediction in order to perform entropy-controlled voltage-frequency scaling, at a sentence granularity, for minimal energy consumption while adhering to a prescribed target latency.

As Transformers are being extended to vision<sup>144,29</sup> and speech<sup>86</sup> applications, it would not be too far-fetched to apply our proposed energy minimization methodology to these tasks as well. We also note that the neural front-end of these Transformer-based networks often comes in the form of embeddings, which can be reused and shared across different finetuned tasks. Given these embedding parameters tend to have a one-time write cost, we can avoid the energy and latency costs of reloading embedding vectors from off-chip memory for different tasks by storing them in embedded non-volatile memories such as ReRAMs or PCMs.

A limitation of this work is that the entropy-based algorithms, so far, work only on encoder-based Transformer architectures such as BERT. Future research will be investigating how to efficiently extend early exit or entropy-based latency reduction capabilities on decoder-based Transformers (such as GPT).

# 6

## CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Chip Learning

HOW TO ARCHITECT AN ON-DEVICE ML TRAINING SYSTEM USING EMBEDDED DYNAMIC MEMORIES? The looming deluge of data is expected to shift AI-related workloads to edge devices, and those workloads will increasingly comprise training<sup>119</sup>. Edge processing has many appealing fea-

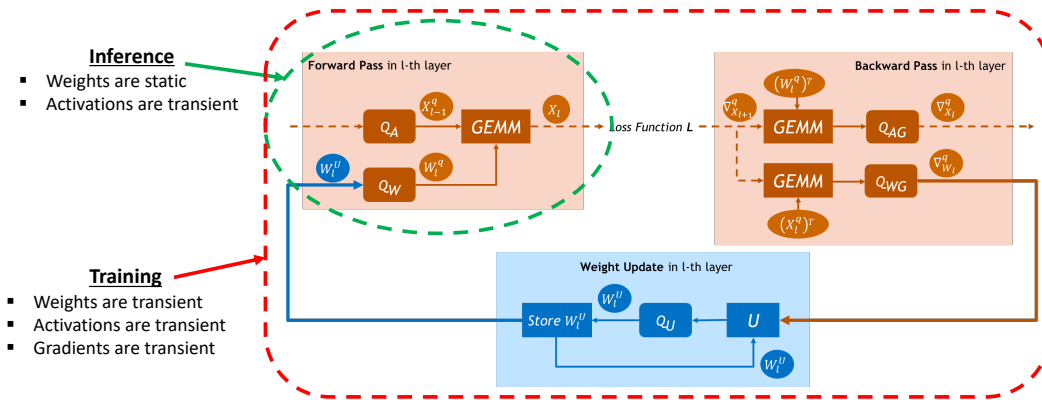


Figure 6.1: Computation requirements during DNN inference and training, highlighting data transience.

tures. It provides advantages in latency, energy efficiency, security, privacy, and autonomy. In particular, there is a growing demand for training DNNs locally within the edge device. For example, Federated Learning (FL) requires on-device DNN training with local user data, which enables users to adapt the DNN model to their personal data and continuously improve their accuracy based on users' preference.

However, training DNNs locally on edge compute platforms present several acute challenges. Compared with the inference operation, DNN training requires extra computations to produce activation gradients and weight gradients, making the computing workload far exceed the inference scenario. Additionally, as also indicated in Figure 6.1, all the intermediate activations during the forward pass need to be buffered in memory for gradient computations, resulting in a significant memory footprint. The limited on-chip memory in edge AI chips often compels an overdependence on costly (high energy and long latency) accesses to off-chip memories.

This problem often puts a constraint on the inference problem size, preventing the deployment of large, state-of-the-art DNNs on mobile platforms. To address this challenge, there has been SoC implementations using compute-in-memory SRAM cells<sup>128,153,32</sup>, and leveraging on-chip non-volatile RRAM macros<sup>106,155</sup> – for the ultimate goal of reducing off-chip memory accesses. Non-

volatile memories (NVMs) such as RRAMs are appealing in the context of inference because they can provide high-density, long-term storage of read-only weights parameters. However, during the training regime, most of the data being used for computation are transient, i.e. their values are updated frequently, as highlighted again in Figure 6.1. Therefore, NVMs may not be a viable solution for edge training as the transitory nature of the data would be testing NVM write endurance constraints. Furthermore, due to the additional requirement of computing activation gradients, weight gradients, as well as weight updates, the amount of transient data during training far exceeds (by more than  $3\times$ ) the volume of static or transient data during inference. Therefore, while the use of SRAMs or compute-in-memory SRAMs may be fitting in the inference scenario, there may not be enough on-chip SRAM capacity to store the large amount of transient data occurring during the training regime.

Given there is a central need for data to be stored temporarily, we propose using embedded DRAM (eDRAM) as the main on-chip storage medium of training data.

#### 6.1 USING eDRAMs AS THE MAIN STORAGE MEDIUM FOR ON-DEVICE TRAINING

eDRAM, in many ways, would be a better fit for this problem. First, there is an opportunity to co-design the DNN accelerator such that the transient data can be stored in eDRAM cells without or with very few occurrences of refresh, which will improve the overall the system energy consumption. Secondly, eDRAM already provides a tangible capacity improvement (at least  $2\times$  compared to SRAMs<sup>38</sup>). Moreover, eDRAM also offers much lower leakage power than SRAM (around  $3.5\times$  according to prior work<sup>18</sup>). The benefits of eDRAM make it an attractive option for DNN computing tasks<sup>145,15</sup>. And, there is an opportunity to further improve DRAM capacity by enabling multiple-level storage which is not possible with SRAMs. This makes eDRAM practical for storing the large volume of training data (activations, weights, and gradients).

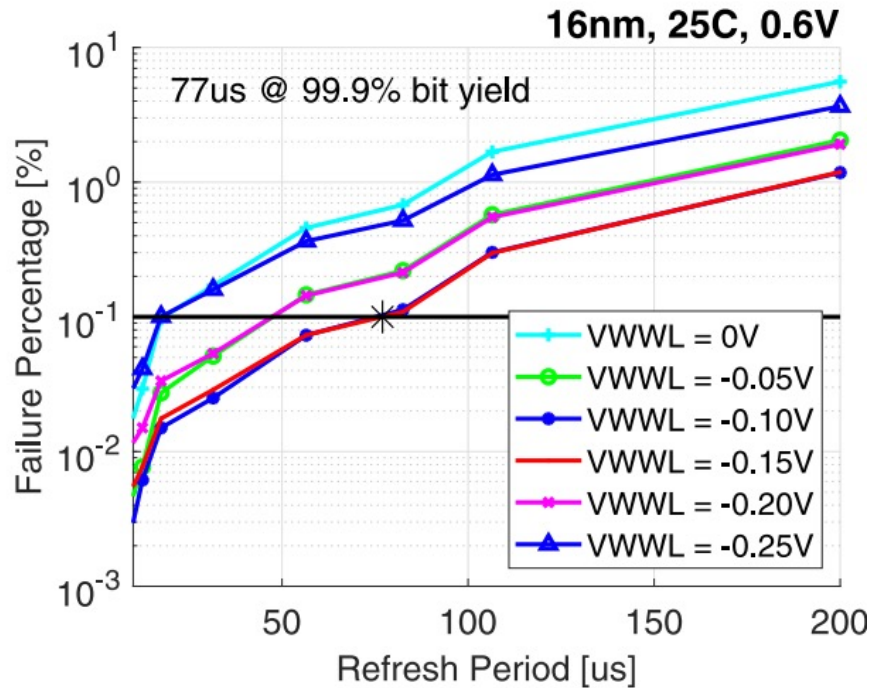
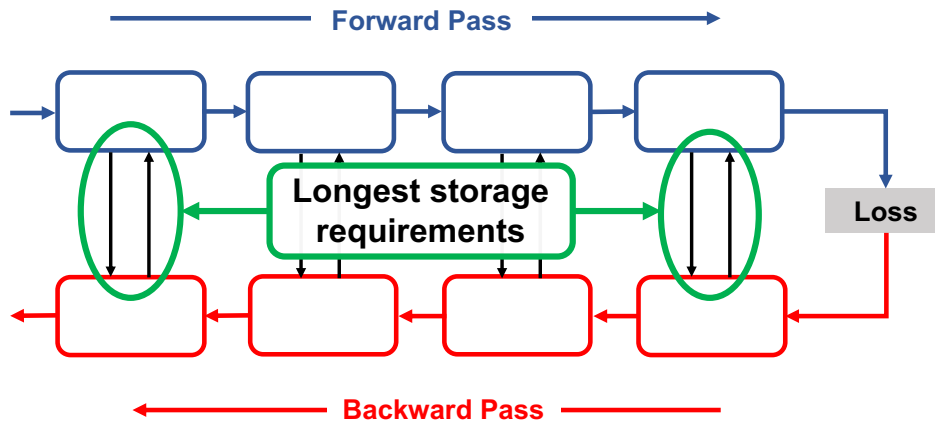


Figure 6.2: eDRAM retention time failures versus refresh period for different write word line voltages (VWWL). Source: R. Giterman et al.<sup>38</sup>

**Challenges:** eDRAM data is stored as a charge on a capacitor that will leak over time, which requires the eDRAM cells to be refreshed periodically for data intactness. Figure 6.2 shows the bit failure percentages under different write word line voltages (VWWL) at a PVT corner of TT/0.6V/25c in 16nm process. A retention time of 77us is measured to achieve a 99.9% bit yield.

It is, therefore, of great importance to inhibit prolonged data storage and reduce the eDRAM refresh frequency. Unfortunately, the backward pass operation requires the activations in the early layers to be stored for a much longer time than later layers (Figure 6.3). In particular, the input activations of the first layer need to be buffered for the entire round trip iteration as they will be absorbed during the two GEMMs of backward pass. Moreover, the weight update of the last layer



**Figure 6.3:** First and last layers in a deep neural network exact the longest storage durations during training, making eDRAM adoption challenging.

needs to be held and consumed during the forward pass GEMM of the next iteration.

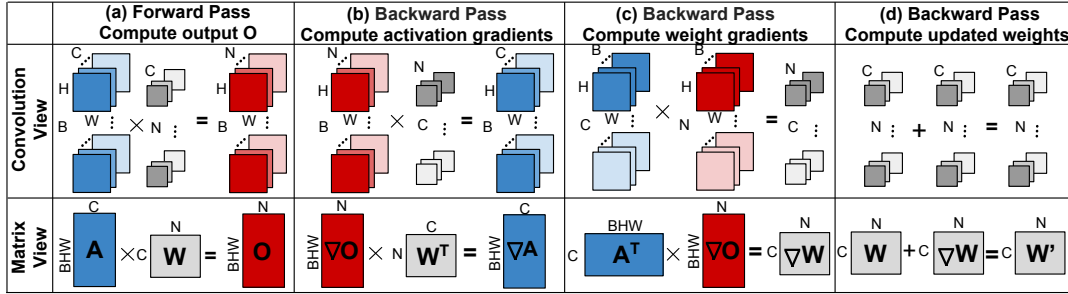
We address this challenge by proposing an algorithm-hardware co-design framework dubbed CAMEL<sup>165</sup>, which leverages recent advances in reversible deep neural network<sup>40</sup> and co-designs the DNN architecture with the hardware system to significantly shorten eDRAM data lifetime during ML training. Furthermore, a 2D Block Floating-Point (BFP) data type reduces the amount of computation per unit area, wherein the higher throughput translates to short data lifetime requirements and helps avoid eDRAM retention failures.

In the later sections, we provide greater details on these aforementioned solutions, which promote efficient ML training with refresh-free eDRAMs. We begin by first introducing the matrix operations within the DNN training process.

## 6.2 COMPUTATIONS IN DNN TRAINING

Each iteration of DNN training consists of a forward pass and a backward pass. During the forward pass, the mini-batch will enter the DNN to compute the training loss. During the backward pass, the gradients are calculated from the training loss, which will then be used to update the DNN





**Figure 6.4:** The forward and backward pass steps for a single layer of DNN training represented in a convolution view and a matrix view. The kernel size of the weight filters is assumed to be  $1 \times 1$  for illustration simplicity.

weights. Figure 6.4 describes the tensor operations and their equivalent matrix computations involved during the forward and backward passes for a convolutional (CONV) layer. Specifically, during the forward pass, the input activations ( $A$ ) are convolved with the layer weights ( $W$ ) to produce the output  $O$ . Then  $O$  will pass through the batch normalization function and activation function, the intermediate results will be used as the input for the later layers. The backward pass computations involve two operations, given the output gradient  $\nabla O$ , it first convolves with the DNN weights  $W$  to generate the input gradient  $\nabla A$ . After that, the output gradient will convolve with the input activation  $A$  to produce the weight gradient  $\nabla W$ . The weight gradient will multiply with the learning rate  $\eta$  and sum with the original weight  $W$  in an elementwise fashion to produce the updated weights  $W'$ . Additional operations are required for other types of optimizers such as Adam<sup>69</sup>.

### 6.3 REVERSIBLE DNN ARCHITECTURES

Reversible Residual Networks (RevNet)<sup>40</sup> is a variant of the residual neural network (ResNet)<sup>49</sup>. RevNet consists of multiple blocks that are reversible in the sense that the input activation of the block can be computed using its output activation. Figure 6.5 shows the architecture of a reversible block.  $F_1$  and  $F_2$  denote a series of DNN layers. For example, for CNN-based architectures, both  $F_1$

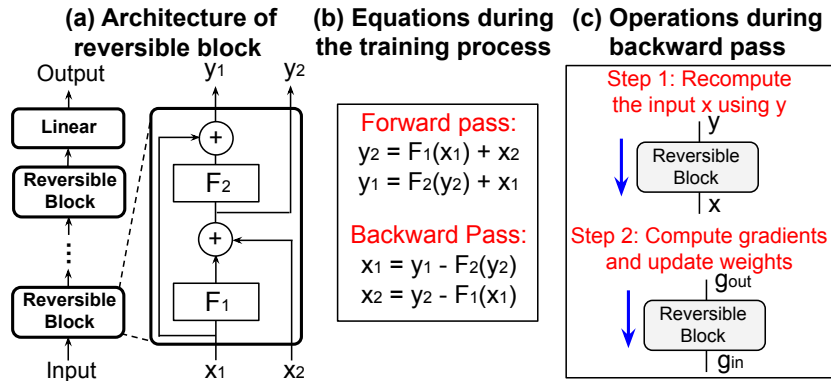


Figure 6.5: Reversible DNN architecture and computations.

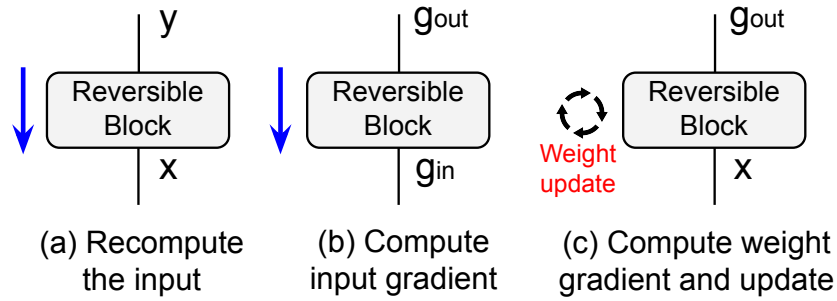


Figure 6.6: Backward pass computation for reversible DNN.

and  $F_2$  consist of a CONV layer, a batch normalization layer and a ReLU layer. The reversible block takes two input activation  $x_1$  and  $x_2$  and produces two output activations  $y_1$  and  $y_2$ . Specifically,  $y_1$  and  $y_2$  can be computed as:

$$y_2 = x_2 + F_1(x_1) \text{ and } y_1 = x_1 + F_2(y_2) \quad (6.1)$$

To recompute  $x_1$  and  $x_2$  with  $y_1$  and  $y_2$ , the following computation can be performed:

$$x_1 = y_1 - F_2(y_2) \text{ and } x_2 = y_2 - F_1(x_1) \quad (6.2)$$

The reversible architecture enables the backward pass computations to be performed without

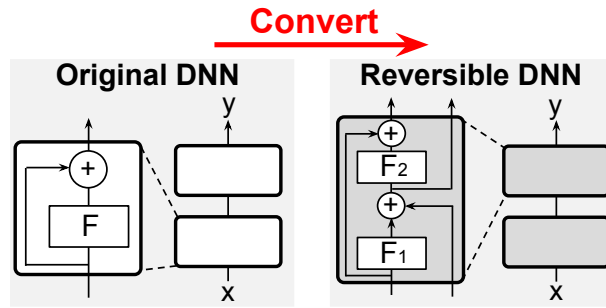
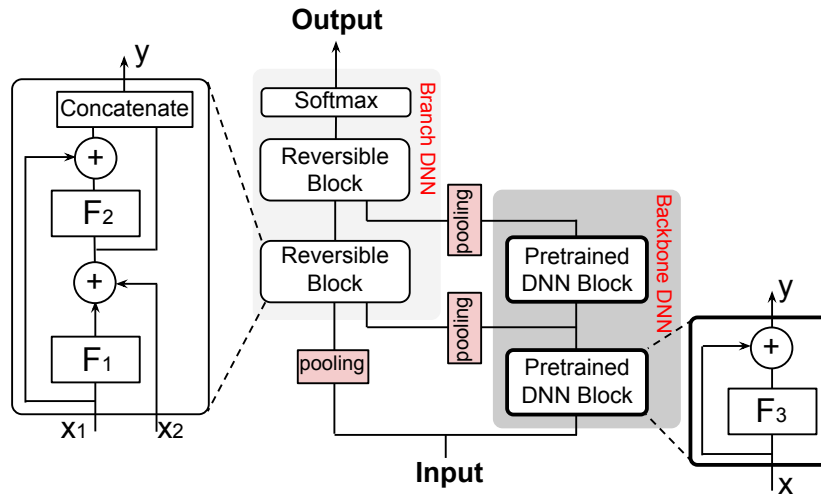


Figure 6.7: Converting a standard DNN into reversible DNN.

the need to store the input activations, as all the input activations  $x$  can be recomputed using the output  $y$ . This further leads to remarkable memory savings and significant reduction on data lifetime. Figure 6.6 depicts the operations involved during the backward pass of a reversible DNN layer. Given the output  $y$ , the input activations are first recomputed with equation 6.2 (Figure 6.6 (a)). After that, the input gradient and weight gradient can be computed with the standard backward pass operations (Figure 6.6 (b,c)).

Any DNN that adopts the residual architecture (e.g., ResNet or transformer) can be easily made reversible. Figure 6.7 depicts an example of a two-layer DNN with residual architecture, where  $F$  represents a stack of layers within each residual block (e.g., CONV+BN+ReLU in ResNet), it can be made reversible by plugging in a scaled version of  $F$  (i.e.,  $F_1, F_2$ ) into the architecture shown in Figure 6.5. In the evaluation section, we demonstrate that the reversible DNN is capable to attain accuracy comparable to that of the standard DNN for various types of  $F$ .

The reversibility feature eliminates the requirement for intermediate activation storage, enabling significant reductions on memory footprint and data lifetime. However, the improved memory performance comes at the price of growth on computing workload due to the recomputation. Worse still, training a reversible DNN from scratch usually takes tens of thousands of iterations to converge, which further exacerbates the latency and energy consumption. One way to solve this problem is reducing the sizes of the DNN, but this will inevitably degrade the accuracy. Next, we



**Figure 6.8:** The architecture of DuDNN, which consists of a two-layer backbone DNN and a two-layer branch DNN that consists of reversible block shown in Figure 6.5. For illustration simplicity, we concatenate outputs by making  $y = [y_1, y_2]$ . We assume the backbone DNN uses residual architecture in this example, but other architectures can also be easily integrated.

describe a family of novel DNN architectures to mitigate this problem.

### 6.3.1 DUPLEX DNN FOR FAST ON-CHIP LEARNING

The proposed DNN architectures, termed *Duplex DNN (DuDNN)*, is shown in Figure 6.8. It consists of a *Branch DNN* (shown in light grey in Figure 6.8) and a *Backbone DNN* (shown in dark grey in Figure 6.8). The branch DNN contains multiple reversible blocks shown in Figure 6.5, the backbone DNN can be any DNN architecture (e.g., ResNet, Transformer). The backbone DNN are pretrained offline using a large-scale training dataset (e.g., ImageNet). During the on-device training stage, the backbone DNN will transfer its knowledge about the current sample to the branch DNN, the branch DNN will take the transferred information and adjust its parameters quickly to adapt the training samples.

Applying neural reversibility promotes recomputation instead of intermediate storage, therefore lessening the acute constraint on satisfying eDRAM retention time during the on-chip training of

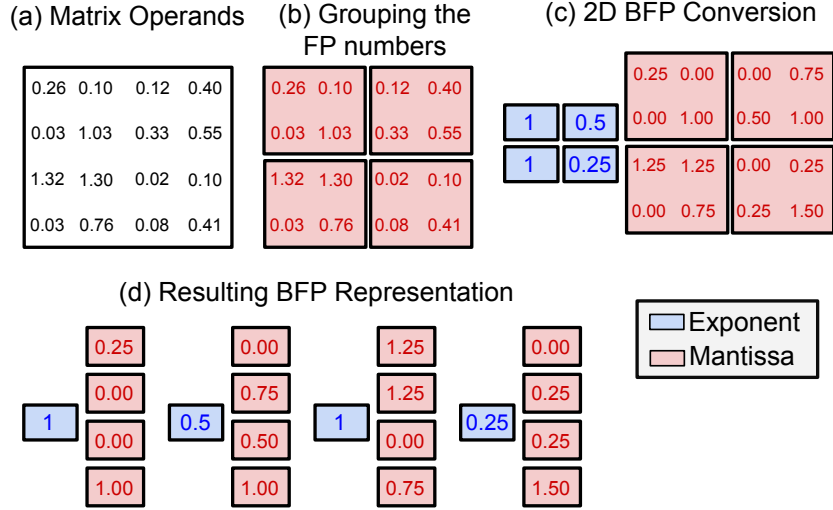
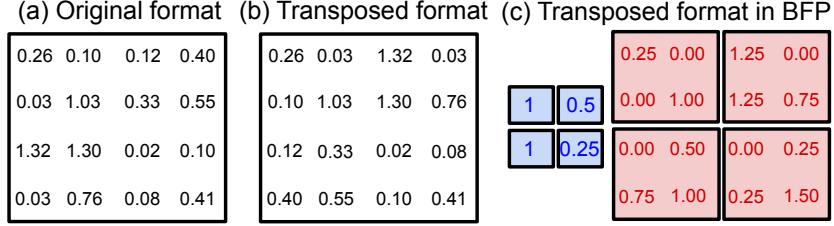


Figure 6.9: 2D BFP quantization on a  $4 \times 4$  matrix. Each group has a size of  $2 \times 2$ .

very deep neural networks. In the next section, we describe a novel number system designed to boost accelerator’s throughput and shorten data lifetime requirements.

#### 6.4 PROPOSING A 2D BLOCK FLOATING-POINT DATATYPE

We apply BFP quantization technique on all the weights, activations, and gradients. During training, each matrix operand is first divided into multiple groups and then quantized with BFP representation. Since the training operation involves the matrix computations under both original and transposed formats (e.g.,  $W$  and  $W^T$ ,  $A$  and  $A^T$  in Figure 6.4), the matrix operands need to be re-quantized with BFP every time the new computation happens, as the transposition will break the group configuration. For example, assume BFP is applied over the weight matrix  $W$  with each BFP group spanning across each column of  $W$ . Let  $Q(W)$  denote the  $W$  in BFP format. Generating  $Q(W^T)$  requires re-quantization over the  $W^T$ , as all the numbers within each column of  $W^T$  is no longer within the same BFP group. These excessive re-quantization operations induce remarkable implementation costs. To mitigate this overhead, we propose a *two-dimensional BFP* (2D BFP)



**Figure 6.10:** (a) The original floating-point matrix. (b) The transposed version of the matrix in (a). (c) The BFP representation of (b), which can be derived easily by transposing the BFP matrix in Figure 6.9 (c).

quantization on the matrix operands. An example is shown in Figure 6.9 where a  $4 \times 4$  floating-point matrix (Figure 6.9 (a)) is divided into four  $2 \times 2$  groups (Figure 6.9 (b)). After that, each group is applied with BFP quantization (Figure 6.9 (c)), resulting in four groups of BFP numbers (Figure 6.9 (d)). Using the 2D BFP representation  $Q(W)$  (Figure 6.9 (c)), we can easily derive the BFP representation  $Q(W^T)$  of its transposed format  $W^T$  (Figure 6.10 (c)) by directly transposing  $Q(W)$  (i.e.,  $Q(W^T) = Q(W)^T$ ), as indicated in Figure 6.10. In this work, we specifically adopt a  $3 \times 3$  2D BFP format with a 4-bit shared exponent field, 5-bit mantissa and 1-bit sign for each of the nine numbers in the BFP group. Therefore, our proposed 2D BFP format contains a total of 58 bits, resulting in an effective precision of 6.4 bits per number.

## 6.5 THE CAMEL HARDWARE ACCELERATOR SYSTEM WITH A HYBRID eDRAM-SRAM MEMORY SUBSYSTEM

To promote greater on-chip training performance and guarantee faithful eDRAM read/write transactions, we design a reconfigurable hardware accelerator system illustrated in Figure 6.11. It consists of a systolic array core and a hybrid eDRAM-SRAM memory subsystem. In this section, we describe the accelerator in greater details, its operational dataflows, as well as, the eDRAM architecture employed for the transient storage of activations and gradients produced during the on-chip training.

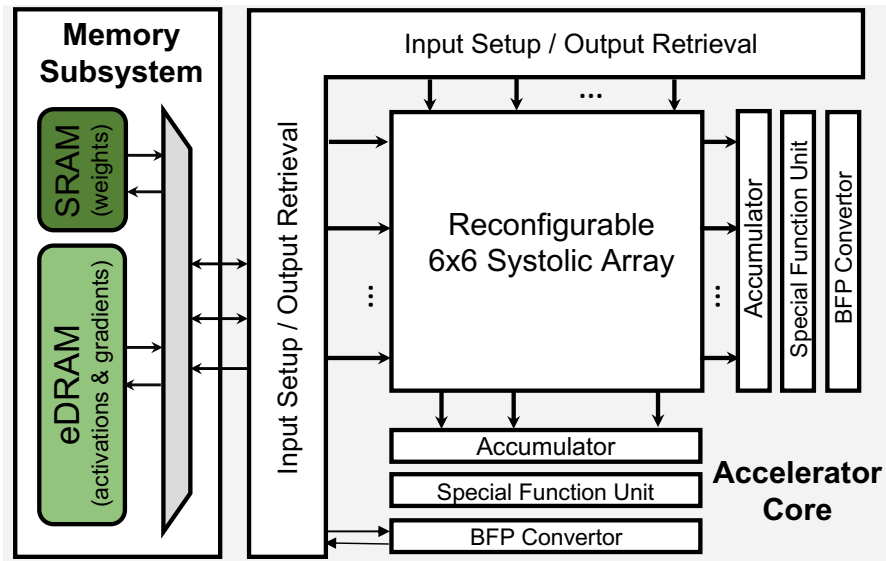


Figure 6.11: The CAMEL training accelerator system.

### 6.5.1 SYSTOLIC ARRAY CORE

The systolic array core consists of a 2D 6-by-6 bidirectional processing element (PE) array that receives staggered inputs from the *I/O* controller and then drains out the computed partial sums onto a post-processing unit composed of the accumulator, special function, and BFP convertor units.

The accelerator benefits from the higher computational accuracy and efficiency of block floating-point encapsulated inside the PE array and the greater hardware density of the fixed-point based post-processing unit. The accumulator aggregates the partial sums into a register file with 64 entries. The special function unit (SFU) computes, in a vectorized fashion, several mathematical and non-linear functions such as ReLU, elementwise addition, subtraction, and multiplication, all of which get invoked during the training passes. The fixed-point outputs of the SFU get converted back into the 2D BFP datatype in order to be stored inside the memory subsystem.

The systolic array core can be reconfigured to operate in three operational workflows as illustrated in Figure 6.12 and described in <sup>162</sup>. During the forward and backward propagation for the

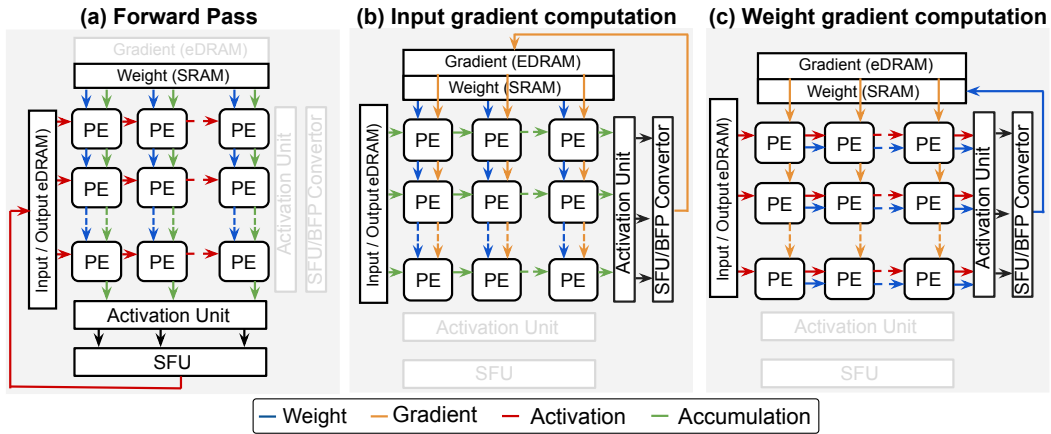


Figure 6.12: Accelerator workflows during the computation of the training passes described in Figure 6.4.

input gradient, the accelerator adopts a weight-stationary dataflow wherein the weights are first pre-loaded from the weight SRAM into the PE weight register and then the inputs are streamed into the array in a staggered cadence. During the forward propagation, activations are fed to the PE array from left to right and the partial sums are accumulated from top to bottom (Figure 6.12 (a)). In contrast, the backward propagation for the input gradient operates in the reverse direction, i.e., the gradient data is streamed from top to bottom while the results are accumulated from left to right (Figure 6.12 (b)). This ensures that matrix-matrix multiplications efficiently operate on the correct dimensionality of the different input operands invoked during the forward and backward passes. Furthermore, the systolic array is configured with an accumulation-stationary dataflow during the backward propagation for the weight gradient. In this mode, the *IO* control unit feeds in the activations and gradients simultaneously into the array and the result accumulates inside the PE until weight gradient tiles are fully computed. At this point, the PEs drain out the accumulated results from left to right for post-processing and weight updating in the SFU, and then the updated weights are stored in the weight SRAM (Figure 6.12 (c)).



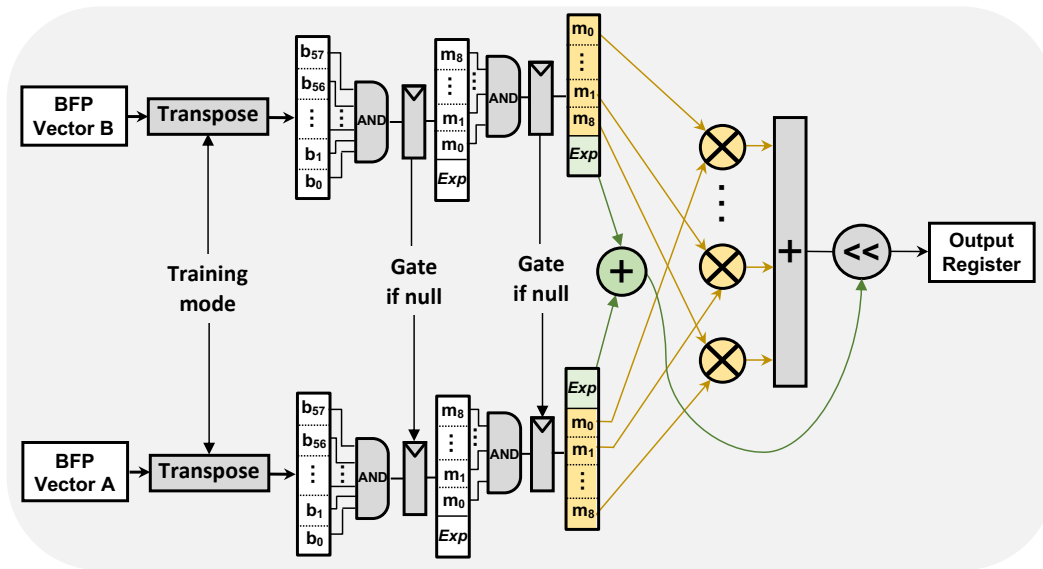
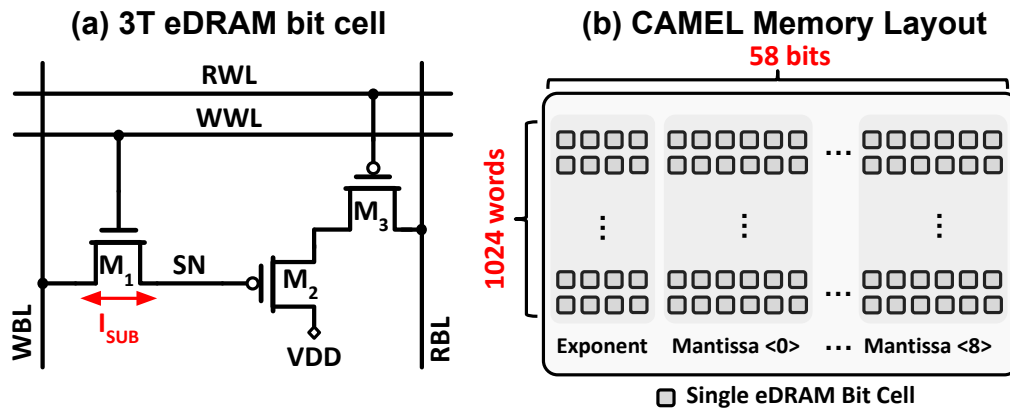


Figure 6.13: Architecture of the CAMEL processing element.

### 6.5.2 BFP PE DESIGN

The architecture of the processing element (PE) of the systolic array is shown in Figure 6.13. The PE receives two tensor operands in the 2D BFP format described in Section 6.4, and performs multiply-and-accumulate (MAC) computations.

Depending on the training mode (i.e., forward or backward pass), the BFP vector gets transposed in place. To promote greater energy efficiency, the PE is equipped with two gating checkpoints before the MAC hardware resources are utilized. First, in case all the 58 bits in one of the two BFP operands are zero, the PE immediately skips to write zero onto the output register. Otherwise, another check is performed on the mantissa bits whereby the mantissa multiplier is gated in case all the mantissa bits in one of the two operands are zero. The BFP MAC executes a *left-shift* operation on the product-sum of the mantissas by the sum of the exponents.



**Figure 6.14:** (a) Schematic view of a 3T eDRAM bit cell, highlighting the main charge/discharge path ( $I_{SUB}$ ) for the storage node (SN). RWL – read word line, RBL – read bit line, WWL – write word line, and WBL – write bit line. (b) Block floating point storage mapping within a  $58 \times 1024$  eDRAM array.

### 6.5.3 HYBRID eDRAM-SRAM MEMORY SUBSYSTEM

The memory subsystem collects and unifies, across channels and filters, the input/output activations and gradients, as well as the master weights. Given the ultimate goal of the training regime is to learn more accurate representations of the master weights, which potentially will be used in subsequent inference tasks, the learned weights are then stored in a static medium (i.e., SRAMs). On the other hand, the transient activation and gradient parameters, which represent the majority of the total training data, are housed in eDRAM memories. The hybrid SRAM-eDRAM memory partitioning provides the benefits of high density storage for the majority of training data and long-term storage for the weights which constitute the final solution of the on-chip learning task. Twelve 32KB eDRAM banks and six 8KB SRAM banks are integrated in the memory subsystem.

### 6.5.4 eDRAM DESIGN

Given there is a central need for the large volume of data to be stored temporarily, eDRAM presents itself as an appealing storage medium for on-chip AI training. An eDRAM structure has the po-

tential to reduce the memory footprint by  $2\times$  with a lower access and leakage energy consumption, compared to SRAM. The main memory design utilized for the CAMEL system evaluation is based on learnings from previous literature<sup>38,95</sup>, which demonstrated full functionality of a 3T eDRAM, even at newer transistor technologies such as FinFET. However, the main drawback of eDRAM is its short retention time and additional energy consumption to refresh previously stored data.

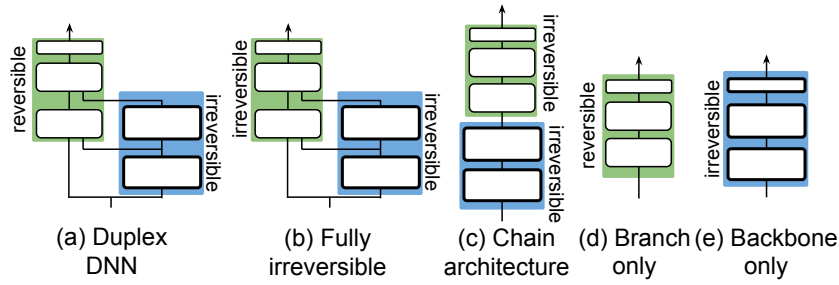
As shown in Fig. 6.14 (a), the main leakage path for the storage node (SN) to charge/discharge is  $I_{SUB}$ , passing through the write transistor  $M_1$ . Therefore, to maximize data lifetime, i.e., extend the memory cell’s retention time, we employ well studied leakage mitigation techniques such as WWL over- and under-drive. Along with co-designing our DNN training algorithm and the hardware to support it, we completely eliminate the need for explicit eDRAM refresh. Fig. 6.14 (b) illustrates the 3T eDRAM array layout for CAMEL. Here, we used memory banks of 58 bits and 1024 words to better match the 2D BFP data shape of our DNN algorithm. We also depict the mapping of BFP into the memory, where we store words comprising one 4-bit shared exponent along with nine 6-bit signed mantissas.

## 6.6 EVALUATION

In this section, we first evaluate the accuracy performance of the reversible DNN, dubbed DuDNN over multiple DNN architectures and datasets. We then provide the hardware evaluation of the CAMEL system.

### 6.6.1 SETTINGS FOR DUPLEX ARCHITECTURE TRAINING

We evaluate the training performance (e.g., classification accuracy) of DuDNN with multiple DNNs over different datasets. We adopt different architectures for the backbone DNN, including CNNs (ResNet-18, ResNet-34, ResNet-50<sup>49</sup>, VGG-16<sup>127</sup>) and vision transformer (ViT)<sup>28</sup>.



**Figure 6.15:** Baseline algorithms for accuracy evaluation. For each baseline architecture, its trainable part is highlighted in green while the frozen part is highlighted in blue.

The backbone DNN is pre-trained offline with the CIFAR-100<sup>74</sup> or ImageNet (ILSVRC 2012)<sup>26</sup> datasets. The pre-trained backbone DNN is then used to guide the training of branch DNN over CIFAR-10<sup>74</sup> or Tiny-ImageNet<sup>63</sup> datasets.

To evaluate the accuracy performance of the DuDNN, we develop some other baseline architectures that are shown in Figure 6.15. The *fully irreversible (FI)* architecture applies irreversible architecture (e.g., residual architecture) over both backbone and branch DNNs. The purpose of this baseline is to evaluate the impact of reversible architecture on validation accuracy. The *Chain architecture (CA)* concatenates the reversible branch DNN with the pre-trained backbone DNN. During the training process, only the reversible component are learnable. The *branch only (BrO)* architecture trains the branch DNN solely without the support from the backbone DNN, BrO can be used to evaluate the impact of the backbone DNN on the convergence behavior of branch DNN. Finally, the *backbone only (BaO)* architecture trains the backbone DNN from scratch.

### 6.6.2 ACCURACY EVALUATION ON DIFFERENT BASELINES

Table 6.1 depicts the DNN test accuracies with different datasets. The names of DNNs indicate the number of layers in branch DNN and backbone DNN. For example, Branch-4 + ResNet-18 denotes a four-layer branch DNN with a ResNet-18 backbone. For BO, the branch DNN is trained from scratch without the backbone. We can see that DuDNN achieves a significantly better accu-

**Table 6.1:** The training accuracies of each method on different DNN architectures.

Architectures	DuDNN		FI		CA		BrO		BaO	
	CIFAR-10	Tiny-ImageNet	CIFAR-10	Tiny-ImageNet	CIFAR-10	Tiny-ImageNet	CIFAR-10	Tiny-ImageNet	CIFAR-10	Tiny-ImageNet
Branch-4 + ResNet-18	87.08%	71.07%	87.20%	71.56%	78.33%	66.30%	55.32%	47.33%	90.32%	90.02%
Branch-5 + ResNet-34	88.81%	74.14%	88.83%	74.88%	79.89%	69.74%	57.80%	49.65%	92.80%	91.98%
Branch-6 + ResNet-50	89.22%	75.88%	89.33%	75.91%	81.14%	70.60%	61.62%	53.09%	93.48%	92.49%
Branch-6 + VGG-16	89.15%	75.98%	89.13%	76.10%	80.98%	70.12%	61.62%	53.09%	92.62%	92.14%
Branch-6 + ViT-12	88.67%	73.56%	88.80%	73.85%	80.72%	68.83%	59.57%	52.36%	91.76%	91.85%

Datasets	DuDNN	FI	CA
MRPC	0.822	0.83	0.787
WNLI	0.606	0.608	0.534
RTE	0.678	0.669	0.612
STS-B	0.879	0.889	0.768
QNLI	0.903	0.909	0.845
SST-2	0.919	0.921	0.868
QQP	0.911	0.911	0.870

**Table 6.2:** Training accuracies over text classification tasks.

racy than CA and BrO on both CIFAR-10 and Tiny-ImageNet datasets, which indicates that the duplex architecture can greatly benefit the branch DNN learning process. DuDNN also achieves a comparable accuracy as FR, which adopts an irreversible architecture in their branch DNN. However, compared with DuDNN, FI and BaO require a tremendous amount of storage to buffer the intermediate activations and further incurs a large overall memory power and latency cost.

We also evaluate DuDNN over text classification tasks with the GLUE dataset<sup>151</sup>, which further includes nine subtasks. We adopt the pretrained 12 layer BERT model as the backbone DNN, and finetune a six-layer Branch DNN, where each reversible block in the branch DNN adopts an attention-based architecture similar to BERT. All the DNNs are trained with 8 epochs. From the results shown in Table 6.2, we observe that DuDNN outperforms CA and achieve a similar performance and FI across all both datasets.

### 6.6.3 HARDWARE SETTINGS OF CAMEL SYSTEM

In this section, we evaluate the hardware performance of the CAMEL system described in Section 6.5. The CAMEL accelerator is designed in synthesizable SystemC and its verilog RTL is generated by the Catapult High-Level Synthesis (HLS) tool<sup>126</sup>. HLS directives are uniformly set with the goal to maximize throughput on the pipelined design. During the HLS process, the SRAM and eDRAM banks are mapped to bit-accurate verilog templates. The following evaluation results are measured at 500MHz using a commercial 16nm process node.

The CAMEL system contains a  $6 \times 6$  systolic array, where each systolic cell can perform a dot product between two  $3 \times 3$  BFP group within one cycle. All the intermediate activations and gradients are stored inside the eDRAM, while the weights are saved in the SRAM.

### 6.6.4 RETENTION TIME PERFORMANCE

To model the behavior of eDRAM, we simulated a  $58 \text{ bits} \times 1024 \text{ words}$  3T eDRAM array to extract retention time and energy consumption metrics. To generate an accurate retention time, we ran 1000 Monte Carlo on-die variation points at the typical process corner using  $VDD = 0.8V$ , and at temperatures ranging from  $-30$  to  $100$ . We measure retention time right after a write operation and until the sense amplifier is still able to correctly read the stored value at 99% yield. In our retention time simulation, we consider a worst-case memory access activity of 0.5 by toggling write bit line (See Figure 6.14) at the system operating frequency, in order to match the high memory access activity of CAMEL.

Figure 6.16 shows that the worst-case retention time ranges from  $3.4\mu s$  at  $100$  to  $30\mu s$  at  $-30$ . We adjust the size of the DuDNN such that the data lifetime is strictly less than the worst retention time ( $3.4\mu s$ ). As an example, Figure 6.17 (a) shows the maximum data lifetime at each layer of Branch DNN in Branch-6 + ResNet-50 during the training on Tiny-ImageNet. As an exam-

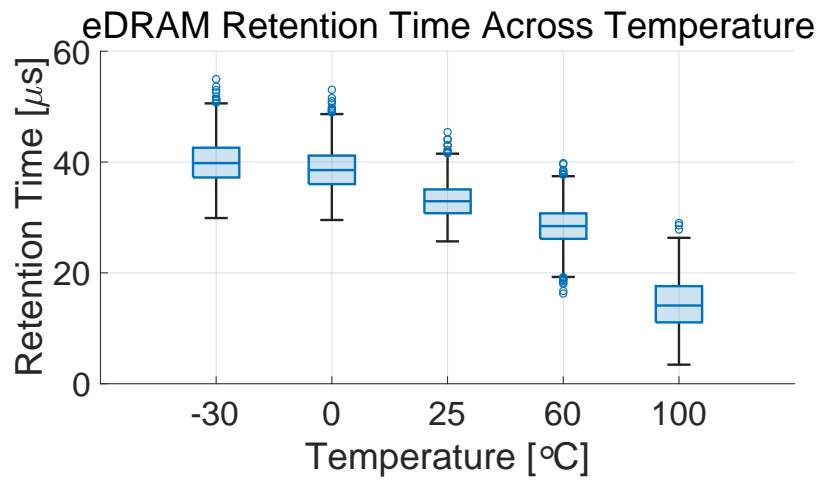


Figure 6.16: 3T eDRAM retention time across temperature.

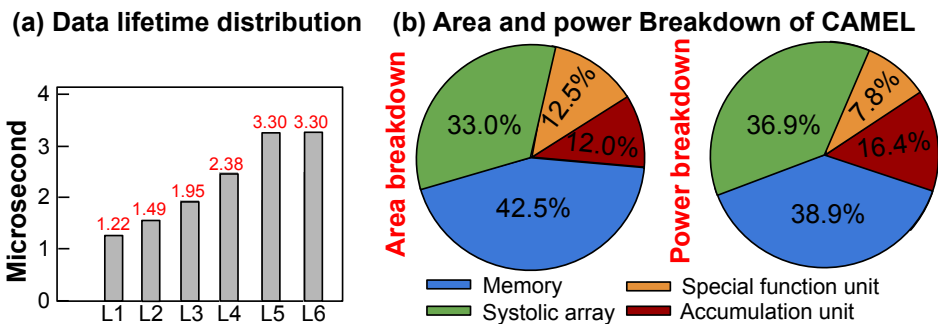


Figure 6.17: (a) Maximum data lifetime at each layer of Branch-6+ResNet-50. (b) Area and power breakdown of CAMEL.

ple, Figure 6.17 (a) shows the maximum data lifetime at each layer of Branch DNN in Branch-6 + ResNet-50 during the training on Tiny-ImageNet.

### 6.6.5 AREA AND POWER BREAKDOWN

Figure 6.17 (b) shows the area and power breakdown of the CAMEL system. Notice that the majority of the area is consumed by the memory subsystem and systolic array, which take 42.5% and 33.0%, followed by the special function unit (12.5%) and accumulator (12.0%). For the power consumption, the memory subsystem still has the largest power consumption (38.9%), followed by the systolic array (36.9%), accumulator (16.4%), special function unit (7.8%).

### 6.6.6 TRAINING PERFORMANCE OF THE CAMEL HARDWARE SYSTEM

In this section, we evaluate the hardware performance of CAMEL system described in Section 6.5 by comparing against another DNN training system implemented without eDRAM.

We implement the DuDNN models that generate the accuracies on Tiny-ImageNet shown in Table 6.1. All the intermediate activations and gradients of DuDNN can totally fit on the on-chip eDRAM, which eliminates the off-chip memory traffic. We use Time-to-Accuracy (TTA)<sup>19</sup> as the evaluation metric to compare the different approaches. TTA measures the amount of total latency required to train the DNN model until reaching a target validation accuracy. In addition, we also measure the total amount of energy during this DNN training process, and name the resulting metric *Energy-to-Accuracy* (ETA).

Figure 6.18 (a) shows the TTA of the baseline systems with different DNN architectures on Tiny-ImageNet. We set the target accuracy to 70%. The training time is normalized by the performance of DuDNN+CAMEL which achieves the smallest TTA. For some of the settings for CA+CAMEL and all settings for BO+CAMEL, their accuracies can not reach the target 70% accu-



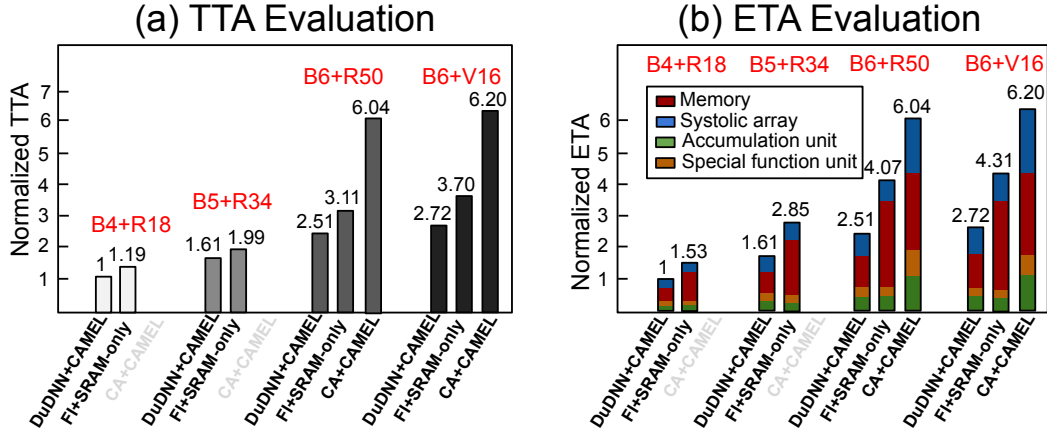


Figure 6.18: TTA and ETA evaluation over different DNN architectures on Tiny-ImageNet. "B", "R" and "V" denote Branch, ResNet and VGG, respectively.

racy, and therefore they are eliminated from the figure. We set the mini-batch size to 48 for all the systems. The DuDNN + CAMEL achieves the optimal performance over other baseline systems. Compared with DuDNN + CAMEL, FR+SRAM-only adopts the SRAM to store the activations and gradients, therefore the memory subsystem consumes a larger chip area than CAMEL, resulting in a smaller systolic array and lower throughput. Despite that DuDNN training includes an extra process to recompute the input, it still ends up with a 20% smaller TTA on average. Lastly, CA+CAMEL and BO+CAMEL (not shown) incur much larger TTAs because of the inferior convergence behavior of CA and BO, which further leads to a large number of iterations for reaching the target accuracy.

Figure 6.18 (b) shows the ETA evaluation of the different approaches. We observe that DuDNN+CAMEL achieves more than  $2\times$  lower ETA on average compared with the other approaches. This is partially due to the fact that DuDNN+CAMEL obtains the smallest TTA among all the systems and further leads to the smallest energy consumption. Additionally, the usage of eDRAMs in CAMEL also induces a much lower memory dynamic power than SRAM-only systems, this can be easily observed with the power breakdown shown in Figure 6.18 (b), where the majority of the energy consumption

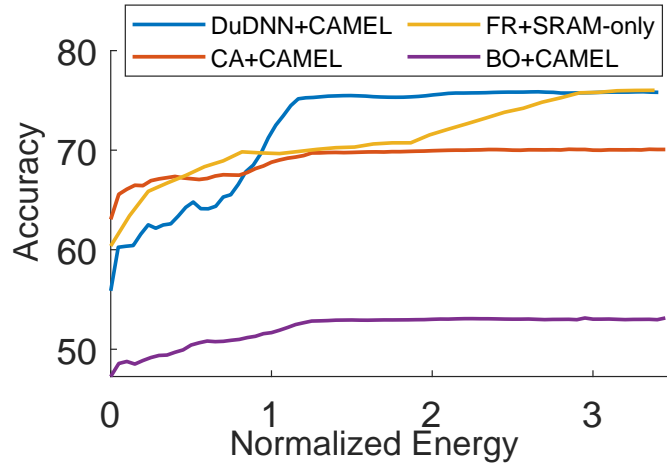


Figure 6.19: Accuracy versus energy.

in FR+SRAM-only is from the memory subsystem.

### 6.6.7 ACCURACY PERFORMANCE WITH ENERGY

Figure 6.19 shows the changes on accuracies with the energy consumption for the different approaches with Branch-6+ResNet-50 architecture on Tiny-ImageNet dataset. We see that DuDNN+CAMEL converges with the least amount of energy. By comparison, FR+SRAM-only ends up with a slightly higher accuracy than DuDNN+CAMEL at a price of much larger total energy consumption. Finally, CA+CAMEL and FO+CAMEL converge at much lower accuracy with much higher energy consumption.

### 6.6.8 IMPACT ON SYSTOLIC ARRAY SIZE

Finally, we investigate the impact on the systolic array size on the data lifetime. Specifically, we increase the systolic array size as well as the number of eDRAM and SRAM banks proportionally for supporting the systolic array operation. Table 6.3 presents the average data lifetime during the training process of Branch-6+ResNet-50 on Tiny-ImageNet. We observe that the data lifetime decreases

Array size	Data Lifetime
$6 \times 6$	$1\times$
$10 \times 10$	$0.90\times$
$12 \times 12$	$0.37\times$

**Table 6.3:** Impact of systolic array size on data lifetime during the training of Branch-6+ResNet-50. The data lifetime are normalized.

sub-linearly as the systolic array size grows. This is because a larger systolic array may also incur a lower utilization rate for some layers of DuDNN, resulting in the same retention time as the small systolic array.

## 6.7 CONCLUSION AND FUTURE WORK

In this work, we propose a novel DNN training system that uses eDRAM as the major storage medium. To mitigate the expensive data refresh in eDRAM, we further propose a novel Duplex DNN architecture that enables a significantly reduced data lifetime during the DNN training process and further eliminates the needs for eDRAM refreshing. Finally, we propose an optimal computation pattern for Duplex DNN training with minimized memory consumption and data lifetime. This full-stack optimization allows for a more than  $2\times$  savings on energy consumption for training different DNN architectures. To our best knowledge, this is the first work that proposes a model-scheduler-architecture stack for efficient on-chip training with refresh-free eDRAMs, which opens up interesting future avenues in a promising direction of research. In particular, a 16nm chip tapeout validating this cross-stack co-design will soon be demonstrated. These results motivate investigating ways to further improve eDRAM capacity via multi-level cell storage, not possible with the conventional 6T SRAM architecture.

# 7

## Conclusion

In this dissertation, we have proposed and discussed four co-designs to promote greater arithmetic density, compute flexibility, and energy efficiency for on-chip deep learning, while prototyping and validating these deep co-designs in silicon systems.

In Chapter 3, we presented *AdaptivFloat* which is a floating-point based mathematical blueprint that uses a configurable exponent bias to enable highly-accurate, low-precision computations without compromising validation accuracy.

In Chapter 4, we presented a 16nm SoC for noise-resistance on-device speech-to-text using a reconfigurable hardware accelerator for efficient computations of attention-based RNNs, dubbed FlexASR, as well as, a Bayesian engine for on-chip real-time denoising. In this work, we showed that we can obviate the need to compute the inference of large ASR neural networks by plugging in the right algorithmic ingredient in the front-end execution pipeline of a monolithic system-on-chip – and doing so without compromising the end-to-end task accuracy.

In Chapter 5, we describes a principled algorithm-hardware co-design solution, validated in a 12nm chip tapeout, that accelerates Transformer workloads by tailoring the accelerator’s latency and energy expenditures according to the complexity of the input query it processes. For this purpose, we use the entropy function as a key metric informing fine-grained, per-sentence latency and power optimizations.

In Chapter 6, we discussed an eDRAM-based fully-on-chip DNN training methodology, prototyped in a 16nm chip tapeout. We showed that by using a combination of reversible neural networks, a custom 2D block floating-point data type, and a systolic-array -based ML training hardware architecture, we can meet eDRAM retention time constraints and avoid refresh penalties.

## 7.1 FUTURE RESEARCH DIRECTIONS

The future of computing in the post-Moore era will be characterized by massive 2D and 3D system integration aimed at solving some of the challenges in zetta-scale computing and beyond; notably interconnect and I/O bandwidth, as well as memory capacity. At the same time, a shift away from compute-centric and towards data-centric processing will accelerate the preeminence of specialized computing, bringing forth more personalized experiences to everyday users. To realize sustainable improvements in energy efficiency, newer algorithms, number formats, synthesized compilers, and computing paradigms will need to be created to lessen power consumption costs that will scale with

increased compute density and scalability. On these lines, here are some follow-on research directions building from this dissertation:

1. **Architectures for Emerging Numerics and Compute-Intensive Applications.** As the demand for greater arithmetic density and computational accuracy grows, number systems will play an increasingly pivotal role in the design of next-generation computing platforms. On one hand, to exact cohesion among programming software interfaces, some newly-created and highly-adopted low-precision number formats (e.g., E4M3, E5M2) will require standardization as chartered by the *IEEE Working Group on Arithmetic Formats for Machine Learning*. On the other hand, I see vast opportunities to break from established standards by sculpting numerics that generate stronger value-add to emerging compute-intensive applications. Building on my recent related work on numerics<sup>134,88,136</sup>, I envision the algorithm-architecture co-design of novel number systems that provide greater amortization of the PPA costs that scale with arithmetic density, and most importantly, without compromising on computational accuracy.
2. **Resilient Multi-level Embedded DRAM Memory Systems.** Achieving high on-chip memory density is necessary to eliminate the significant inefficiencies stemming from off-chip memory accesses. Building on my current eDRAM research and tapeout learnings, there is an opportunity to further improve eDRAM capacity by enabling multi-level cell (MLC) storage. This will make eDRAM practical for storing the large volume of training data generated by more sophisticated DNNs. However, the drawback of MLC storage is that the number of faults increases with the number of levels, and the refresh requirement also becomes stricter. I see opportunities to develop custom MLC eDRAM topologies, newer runtime schedulers, and frameworks for eDRAM-based approximate computing, as well as, MLC eDRAM-based chip tapeouts – all with the goal of achieving a high degree of algorithmic

resilience, storage and energy efficiency.

3. **Cryogenic CMOS Circuits for Quantum Readout and Control.** A major advantage of cryogenic CMOS circuits is their ultra-low power consumption, which is crucial for maintaining the low temperatures required for superconducting qubits to operate. When properly designed, they also offer high speed and high integration density, which would be essential for scaling up quantum systems to larger sizes. Recent work<sup>37</sup> demonstrates the viability of 2T-based gain-cell eDRAMs under cryogenic conditions, revealing significant advantages in area (-73%), leakage power (-97%), retention power (-76%), and energy (-66%) when compared to conventional 6T-SRAM. This is a promising line of device research that, in my view, offers a lot more room for PPA improvement.
4. **Smart Power Management.** The problem of energy efficiency still exists, and in fact, has been made worse by the unrelenting pursuit of omniscient and omnipotent AI. Building on my past research on entropy-based early exit and DVFS<sup>135,131</sup> for inferencing large language models, I envision future AI-aided smart power management schemes and circuits providing finer-grained control on power and latency consumption in next-generation computing platforms. It is increasingly critical to integrate mixed-signal architectures and software drivers that are tweaked to interface with the AI algorithm in order to promote high system energy efficiency across all operational modes.
5. **Scalable Silicon Systems.** The waning effectiveness of transistor scaling is motivating an industry-wide push towards modularity and heterogeneous integration. I had a front row seat in the engineering of this trend having designed high-speed communication circuits customized for the Embedded Multi-Die Interconnect Bridge (EMIB) in Intel's first products adopting this chip-to-chip packaging technology. We are now seeing newer solid-state solutions aiming to democratize chip-to-chip scalability. In particular, UCIE<sup>123</sup>, and Bunch

of Wires<sup>5</sup>, which are open industry standard interconnects, offering high-bandwidth and low-latency on-package connectivity between chiplets.

2D, 2.5D, or 3D chip making should not be an esoteric endeavour. Academia has a huge role to play in lowering the entry barrier for designing high performance scalable systems in a cost-effective and user-friendly manner. In fact, the recent global chip shortage has caused significant micro- and macro- economic consequences, highlighting the urgent need to explore new research avenues that focus on reducing the overall design effort and cost across the software-silicon stack.



## References

- [1] Akhlaghi, V., Yazdanbakhsh, A., Samadi, K., Gupta, R. K., & Esmailzadeh, H. (2018). Sna-  
pea: Predictive early activation for reducing computation in deep convolutional neural net-  
works. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*  
(pp. 662–673).
- [2] Albericio, J., Judd, P., Hetherington, T., Aamodt, T., Jerger, N. E., & Moshovos, A. (2016).  
Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proceedings of the*  
*43rd International Symposium on Computer Architecture*.
- [3] Amatriain, X. (2023). Transformer models: an introduction  
and catalog—2023 edition. [https://amatriain.net/blog/  
transformer-models-an-introduction-and-catalog-2d1e9039f376/](https://amatriain.net/blog/transformer-models-an-introduction-and-catalog-2d1e9039f376/).
- [4] Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper,  
J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G. F., Elsen, E., Engel,  
J., Fan, L. J., Fougner, C., Hannun, A. Y., Jun, B., Han, T. X., LeGresley, P., Li, X., Lin, L.,  
Narang, S., Ng, A., Ozair, S., Prenger, R. J., Qian, S., Raiman, J., Satheesh, S., Seetapun, D.,  
Sengupta, S., Sriram, A., Wang, C.-J., Wang, Y., Wang, Z., Xiao, B., Xie, Y., Yogatama, D.,  
Zhan, J., & Zhu, Z. (2015). Deep speech 2 : End-to-end speech recognition in english and  
mandarin. In *International Conference on Machine Learning*.
- [5] Ardalan, S., Cirit, H., Farjad, R., Kuemerle, M., Poulton, K., Subramanian, S., & Vinnakota,  
B. (2020). Bunch of wires: An open die-to-die interface. In *2020 IEEE Symposium on High-  
Performance Interconnects (HOTI)* (pp. 9–16).
- [6] Arm (2020). Arm machine learning processor. [https://developer.arm.com/ip-products/  
processors/machine-learning/arm-ml-processor](https://developer.arm.com/ip-products/processors/machine-learning/arm-ml-processor).
- [7] Ba, J., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *ArXiv*, abs/1607.06450.
- [8] Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning  
to align and translate. In *3rd International Conference on Learning Representations, ICLR*  
*2015*.

- [9] Banner, R., Hubara, I., Hoffer, E., & Soudry, D. (2018). Scalable methods for 8-bit training of neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18* (pp. 5151–5159). Red Hook, NY, USA: Curran Associates Inc.
- [10] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020a). Language models are few-shot learners. *Advances in neural information processing systems*, 33, 1877–1901.
- [11] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T. J., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020b). Language models are few-shot learners. *ArXiv*, abs/2005.14165.
- [12] Chan, W., Jaitly, N., Le, Q. V., & Vinyals, O. (2015). Listen, attend and spell. *ArXiv*, abs/1508.01211.
- [13] Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., & Temam, O. (2014a). Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '14* (pp. 269–284). New York, NY, USA: ACM.
- [14] Chen, Y., Emer, J., & Sze, V. (2016). Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (pp. 367–379).
- [15] Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., & Temam, O. (2014b). Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 609–622).: IEEE Computer Society.
- [16] Cho, K., van Merriënboer, B., Çaglar Gülçehre, Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*.
- [17] Choi, J., Venkataramani, S., Srinivasan, V. V., Gopalakrishnan, K., Wang, Z., & Chuang, P. (2019). Accurate and efficient 2-bit quantized neural networks. *Proceedings of Machine Learning and Systems*, 1, 348–359.
- [18] Chun, K. C., Jain, P., Lee, J. H., & Kim, C. H. (2011). A 3t gain cell embedded dram utilizing preferential boosting for high density and low power on-die caches. *IEEE Journal of Solid-State Circuits*, 46(6), 1495–1505.

- [19] Coleman, C., Narayanan, D., Kang, D., Zhao, T., Zhang, J., Nardi, L., Bailis, P., Olukotun, K., Ré, C., & Zaharia, M. (2017). Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101), 102.
- [20] Courbariaux, M. & Bengio, Y. (2016). Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830.
- [21] Courbariaux, M., Bengio, Y., & David, J.-P. (2014). Training deep neural networks with low precision multiplications. *arXiv: Learning*.
- [22] Courbariaux, M., Bengio, Y., & David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*.
- [23] Dai, S., Venkatesan, R., Ren, H., Zimmer, B., Dally, W. J., & Khailany, B. (2021). Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *ArXiv*, abs/2102.04503.
- [24] Darvish Rouhani, B., Lo, D., Zhao, R., Liu, M., Fowers, J., Ovtcharov, K., Vinogradsky, A., Massengill, S., Yang, L., Bittner, R., Forin, A., Zhu, H., Na, T., Patel, P., Che, S., Chand Koppaka, L., SONG, X., Som, S., Das, K., T, S., Reinhardt, S., Lanka, S., Chung, E., & Burger, D. (2020). Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems*, volume 33 (pp. 10271–10281): Curran Associates, Inc.
- [25] Dean, J., Patterson, D., & Young, C. (2018). A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro*, 38(2), 21–29.
- [26] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (pp. 248–255): IEEE.
- [27] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.
- [28] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020a). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.
- [29] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020b). An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929.

- [30] Drumond, M. et al. (2018). End-to-end DNN training with block floating point arithmetic. *arXiv*, abs/1804.01526.
- [31] Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., Zoph, B., Fedus, L., Bosma, M., Zhou, Z., Wang, T., Wang, Y. E., Webster, K., Pellat, M., Robinson, K., Meier-Hellstern, K. S., Duke, T., Dixon, L., Zhang, K., Le, Q. V., Wu, Y., Chen, Z., & Cui, C. (2021). Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*.
- [32] Eckert, C., Wang, X., Wang, J., Subramaniyan, A., Iyer, R., Sylvester, D., Blaauw, D., & Das, R. (2018). Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture, ISCA '18* (pp. 383–396).: IEEE Press.
- [33] Fedus, W., Zoph, B., & Shazeer, N. (2021). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res*, 23, 1–40.
- [34] Fowers, J., Ovtcharov, K., Papamichael, M., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., Adams, L., Ghandi, M., Heil, S., Patel, P., Sapek, A., Weisz, G., Woods, L., Lanka, S., Reinhardt, S. K., Caulfield, A. M., Chung, E. S., & Burger, D. (2018). A configurable cloud-scale dnn processor for real-time ai. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)* (pp. 1–14).
- [35] Freund, K. & Moorhead, P. (2020). The graphcore second generation ipu.
- [36] Fujiki, D., Wang, X., Subramaniyan, A., & Das, R. (2021). *In-/Near-Memory Computing*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers.
- [37] Garzón, E., Greenblatt, Y., Harel, O., Lanuzza, M., & Teman, A. (2021). Gain-cell embedded dram under cryogenic operation—a first study. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(7), 1319–1324.
- [38] Giterman, R., Shalom, A., Burg, A., Fish, A., & Teman, A. (2020). A 1-mbit fully logic-compatible 3t gain-cell embedded dram in 16-nm finfet. *IEEE Solid-State Circuits Letters*, 3, 110–113.
- [39] Gomez, A. N., Ren, M., Urtasun, R., & Grosse, R. B. (2017a). The reversible residual network: Backpropagation without storing activations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, volume 30: Curran Associates, Inc.
- [40] Gomez, A. N., Ren, M., Urtasun, R., & Grosse, R. B. (2017b). The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30.

- [41] Google (2019). Bfloat16: The secret to high performance on cloud tpus. Accessed: 2023-03-06.
- [42] Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15* (pp. 1737–1746).: JMLR.org.
- [43] Gustafson, J. et al. (2017). Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*.
- [44] Ham, T. J., Jung, S. J., Kim, S., Oh, Y. H., Park, Y., Song, Y., Park, J., Lee, S.-H., Park, K., Lee, J., & Jeong, D.-K. (2020). A<sup>3</sup>: Accelerating attention mechanisms in neural networks with approximation. *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, (pp. 328–341).
- [45] Ham, T. J., Lee, Y., Seo, S. H., Kim, S., Choi, H., Jung, S. J., & Lee, J. W. (2021). Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)* (pp. 692–705).
- [46] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., & Dally, W. J. (2016). Eie: Efficient inference engine on compressed deep neural network. *SIGARCH Comput. Archit. News*, 44(3).
- [47] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149.
- [48] Hannun, A. Y., Case, C., Casper, J., Catanzaro, B., Diamos, G. F., Elsen, E., Prenger, R. J., Sathesh, S., Sengupta, S., Coates, A., & Ng, A. (2014). Deep speech: Scaling up end-to-end speech recognition. *ArXiv*, abs/1412.5567.
- [49] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 770–778).
- [50] He, Y., Liu, P., Wang, Z., Hu, Z., & Yang, Y. (2018). Filter pruning via geometric median for deep convolutional neural networks acceleration. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 4335–4344).
- [51] Hegde, K., Yu, J., Agrawal, R., Yan, M., Pellauer, M., & Fletcher, C. W. (2018). Ucn: Exploiting computational reuse in deep neural networks via weight repetition. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (pp. 674–687).
- [52] Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

- [53] Horowitz, M. (2014). Computing’s energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (pp. 10–14).
- [54] Huang, B.-Y., Lyubomirsky, S., Li, Y., He, M., Tambe, T., Smith, G. H., Gaonkar, A., Canumalla, V., Wei, G., Gupta, A., Tatlock, Z., & Malik, S. (2022). Specialized accelerators and compiler flows: Replacing accelerator apis with a formal software/hardware interface. *ArXiv*, abs/2203.00218.
- [55] Huang, B.-Y., Zhang, H., Subramanyan, P., Vazel, Y., Gupta, A., & Malik, S. (2018). Instruction-level abstraction (ila): A uniform specification for system-on-chip (soc) verification. *ACM Transactions on Design Automation of Electronic Systems*, 24.
- [56] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.*, 18(1), 6869–6898.
- [57] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A. G., Adam, H., & Kalenichenko, D. (2017). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (pp. 2704–2713).
- [58] Jia, Z., Tillman, B., Maggioni, M., & Scarpazza, D. P. (2019). Dissecting the graphcore ipu architecture via microbenchmarking.
- [59] Johnson, J. (2018). Rethinking floating point for deep learning. *CoRR*, abs/1811.01721.
- [60] Jouppi, N. P., Kurian, G., Li, S., Ma, P., Nagarajan, R., Nai, L., Patil, N., Subramanian, S., Swing, A., Towles, B., et al. (2023). Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. *arXiv preprint arXiv:2304.01433*.
- [61] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., & Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (pp. 1–12).

- [62] Judd, P. et al. (2016). Stripes: Bit-serial deep neural network computing. In *MICRO*.
- [63] Kaggle (2017). Tiny imagenet dataset. <https://www.kaggle.com/c/tiny-imagenet>.
- [64] Keller, B., Venkatesan, R., Dai, S., Tell, S. G., Zimmer, B., Dally, W. J., Thomas Gray, C., & Khailany, B. (2022). A 17–95.6 tops/w deep learning inference accelerator with per-vector scaled 4-bit quantization for transformers in 5nm. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)* (pp. 16–17).
- [65] Khailany, B. et al. (2018). A modular digital vlsi flow for high-productivity soc design. In *DAC*.
- [66] Khoram, S. et al. (2018). Adaptive quantization of neural networks. In *ICLR*.
- [67] Kim, M., Smaragdis, P., Ko, G. G., & Rutenbar, R. A. (2012). Stereophonic spectrogram segmentation using markov random fields. In *2012 IEEE International Workshop on Machine Learning for Signal Processing* (pp. 1–6).
- [68] Kim, S., Hooper, C., Wattanawong, T., Kang, M., Yan, R., Genç, H., Dinh, G., Huang, Q., Keutzer, K., Mahoney, M. W., Shao, Y. S., & Gholami, A. (2023). Full stack optimization of transformer inference: a survey. *ArXiv*, abs/2302.14017.
- [69] Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [70] Klein, G., Kim, Y., Deng, Y., Senellart, J., & Rush, A. (2017). OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations* (pp. 67–72). Vancouver, Canada: Association for Computational Linguistics.
- [71] Ko, G. G., Chai, Y., Donato, M., Whatmough, P. N., Tambe, T., Rutenbar, R. A., Brooks, D., & Wei, G.-Y. (2020a). A 3mm<sup>2</sup> programmable bayesian inference accelerator for unsupervised machine perception using parallel gibbs sampling in 16nm. In *2020 IEEE Symposium on VLSI Circuits* (pp. 1–2).
- [72] Ko, G. G., Chai, Y., Donato, M., Whatmough, P. N., Tambe, T., Rutenbar, R. A., Wei, G., & Brooks, D. (2020b). A scalable bayesian inference accelerator for unsupervised learning. In *2020 IEEE Hot Chips 32 Symposium (HCS)* (pp. 1–27).
- [73] Köster, U., Webb, T. J., Wang, X., Nassar, M., Bansal, A. K., Constable, W., Elibol, O. H., Hall, S., Hornof, L., Khosrowshahi, A., Kloss, C., Pai, R. J., & Rao, N. (2017). Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *NIPS*.
- [74] Krizhevsky, A., Nair, V., & Hinton, G. (2014). The cifar-10 dataset.
- [75] Kung, H. T. (1982). Why systolic architectures? *Computer*, 15(1), 37–46.

- [76] Kung, H. T., McDanel, B., & Zhang, S. Q. (2020). Term quantization: Furthering quantization at run time. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '20*: IEEE Press.
- [77] Kwon, H., Samajdar, A., & Krishna, T. (2018). Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *SIGPLAN Not.*, 53(2), 461–475.
- [78] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942.
- [79] LeCun, Y., Denker, J., & Solla, S. (1989). Optimal brain damage. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems*, volume 2: Morgan-Kaufmann.
- [80] Lee, E. H. et al. (2017). Lognet: Energy-efficient neural networks using logarithmic computation. In *ICASSP*.
- [81] Li, Z., Ghodrati, S., Yazdanbakhsh, A., Esmailzadeh, H., & Kang, M. (2022). Accelerating attention through gradient-based learned runtime pruning. *Proceedings of the 49th Annual International Symposium on Computer Architecture*.
- [82] Liu, D., Chen, T., Liu, S., Zhou, J., Zhou, S., Teman, O., Feng, X., Zhou, X., & Chen, Y. (2015). Pudianna: A polyvalent machine learning accelerator. *SIGPLAN Not.*, 50(4), 369–381.
- [83] Liu, S., Du, Z., Tao, J., Han, D., Luo, T., Xie, Y., Chen, Y., & Chen, T. (2016). Cambricon: An instruction set architecture for neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture, ISCA '16* (pp. 393–405).
- [84] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.
- [85] Lu, L., Jin, Y., Bi, H., Luo, Z., Li, P., Wang, T., & Liang, Y. (2021). Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '21* (pp. 977–991). New York, NY, USA: Association for Computing Machinery.
- [86] Lu, L., Liu, C., Li, J., & Gong, Y. (2020). Exploring transformers for large-scale speech recognition. In *Interspeech*.
- [87] Luo, J.-H. & Wu, J. (2018). Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *ArXiv*, abs/1805.08941.
- [88] Mahmoud, A., Tambe, T., Aloui, T., Brooks, D., & Wei, G.-Y. (2022). Goldeneye: A platform for evaluating emerging numerical data formats in dnn accelerators. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.



- [89] Mahmoud, M., Edo, I., Zadeh, A. H., Mohamed Awad, O., Pekhimenko, G., Albericio, J., & Moshovos, A. (2020). Tensordash: Exploiting sparsity to accelerate deep neural network training. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (pp. 781–795).
- [90] Meinerzhagen, P. et al. (2018). 2.3 an energy-efficient graphics processor featuring fine-grain dvfs with integrated voltage regulators, execution-unit turbo, and retentive sleep in 14nm tri-gate cmos. In *ISSCC*.
- [91] Migacz, S. (2017). 8-bit inference with tensorrt. In *NVIDIA GPU Tech Conf*.
- [92] Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). Pruning convolutional neural networks for resource efficient transfer learning. *ArXiv*, abs/1611.06440.
- [93] Moons, B., Goetschalckx, K., Van Berckelaer, N., & Verhelst, M. (2017). Minimum energy quantized neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers* (pp. 1921–1925).
- [94] Mujawar, M. & Vijaya Saradhi, D. (2022). Design of low-cost active noise cancelling (anc) circuit using ki-cad. In H. S. Saini, R. K. Singh, M. Tariq Beg, R. Mulaveesala, & M. R. Mahmood (Eds.), *Innovations in Electronics and Communication Engineering* (pp. 109–116). Singapore: Springer Singapore.
- [95] Narinx, J., Giterman, R., Bonetti, A., Frigerio, N., Aprile, C., Burg, A., & Leblebici, Y. (2019). A 24 kb single-well mixed 3t gain-cell edram with body-bias in 28 nm fd-soi for refresh-free dsp applications. In *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)* (pp. 219–222).
- [96] Noh, H. et al. (2017). Regularizing deep neural networks by noise: Its interpretation and optimization. In *NeurIPS*.
- [97] NVIDIA (2021). Accelerating ai training with nvidia tf32 tensor cores. Accessed: 2023-03-06.
- [98] NVIDIA (2022). Nvidia a100 80gb pcie gpu.
- [99] NVIDIA (2022). Nvidia hopper architecture in-depth. Accessed: 2023-03-16.
- [100] Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015). Librispeech: An asr corpus based on public domain audio books. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 5206–5210).
- [101] Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S. W., & Dally, W. J. (2017). Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on*

- Computer Architecture*, ISCA '17 (pp. 27–40). New York, NY, USA: Association for Computing Machinery.
- [102] Park, E. et al. (2018). Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *ISCA*.
  - [103] Park, J., Yoon, H., Ahn, D., Choi, J., & Kim, J.-J. (2020). Optimus: Optimized matrix multiplication structure for transformer neural network accelerator. In I. Dhillon, D. Papailiopoulos, & V. Sze (Eds.), *Proceedings of Machine Learning and Systems*, volume 2 (pp. 363–378).
  - [104] Pentecost, L., Donato, M., Reagen, B., Gupta, U., Ma, S., Wei, G.-Y., & Brooks, D. (2019). Maxnvm: Maximizing dnn storage density and inference efficiency with sparse encoding and error mitigation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52 (pp. 769–781). New York, NY, USA: Association for Computing Machinery.
  - [105] Prabhakar, R., Zhang, Y., Koeplinger, D., Feldman, M., Zhao, T., Hadjis, S., Pedram, A., Kozyrakis, C., & Olukotun, K. (2017). Plasticine: A reconfigurable architecture for parallel patterns. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (pp. 389–402).
  - [106] Prabhu, K., Gural, A., Khan, Z. F., Radway, R. M., Giordano, M., Koul, K., Doshi, R., Kustin, J. W., Liu, T., Lopes, G. B., Turbinder, V., Khwa, W.-S., Chih, Y.-D., Chang, M.-F., Lallement, G., Murmann, B., Mitra, S., & Raina, P. (2022). Chimera: A 0.92-tops, 2.2-tops/w edge ai accelerator with 2-mbyte on-chip foundry resistive ram for efficient training and inference. *IEEE Journal of Solid-State Circuits*, 57(4), 1013–1026.
  - [107] Qian, X.-Y. & Klabjan, D. (2021). A probabilistic approach to neural network pruning. In *International Conference on Machine Learning*.
  - [108] Qin, E., Samajdar, A., Kwon, H., Nadella, V., Srinivasan, S., Das, D., Kaul, B., & Krishna, T. (2020). Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (pp. 58–70).
  - [109] Qu, Z., Liu, L., Tu, F., Chen, Z., Ding, Y., & Xie, Y. (2022). Dota: Detect and omit weak attentions for scalable transformer acceleration. *ASPLOS '22* (pp. 14–26). New York, NY, USA: Association for Computing Machinery.
  - [110] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.
  - [111] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners.

- [112] Reagen, B., Gupta, U., Adolf, B., Mitzenmacher, M., Rush, A. M., Wei, G.-Y., & Brooks, D. M. (2017). Weightless: Lossy weight encoding for deep neural network compression. *ArXiv*, abs/1711.04686.
- [113] Reagen, B., Whatmough, P., Adolf, R., Rama, S., Lee, H., Lee, S. K., Hernández-Lobato, J. M., Wei, G.-Y., & Brooks, D. (2016). Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16 (pp. 267–278). Piscataway, NJ, USA.
- [114] Sahu, S. & Roberts, G. (1999). On convergence of the em algorithm and the gibbs sampler. In *Statistics and Computing*.
- [115] Salimans, T. & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*.
- [116] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.
- [117] Sanh, V., Wolf, T., & Rush, A. M. (2020). Movement pruning: Adaptive sparsity by fine-tuning. *ArXiv*, abs/2005.07683.
- [118] Schweber, B. (2018). Active noise cancellation. <https://www.analogictips.com/active-noise-cancellation-part-1-concept-and-principles-faq/>.
- [119] Semiconductor Research Corporation (2020). The decadal plan for semiconductors. Accessed: 2022-11-20.
- [120] Settle, S. et al. (2018). Quantizing convolutional neural networks for low-power high-throughput inference engines. *ArXiv*, abs/1805.07941.
- [121] Sevilla, J., Heim, L., Ho, A. C., Besiroglu, T., Hobbhahn, M., & Villalobos, P. (2022). Compute trends across three eras of machine learning. *2022 International Joint Conference on Neural Networks (IJCNN)*.
- [122] Shao, Y. S., Clemons, J., Venkatesan, R., Zimmer, B., Fojtik, M., Jiang, N., Keller, B., Klinefelter, A., Pinckney, N., Raina, P., Tell, S. G., Zhang, Y., Dally, W. J., Emer, J., Gray, C. T., Khailany, B., & Keckler, S. W. (2019). Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '19 (pp. 14–27). New York, NY, USA: Association for Computing Machinery.
- [123] Sharma, D. (2023). System on a package innovations with universal chiplet interconnect express (ucie) interconnect. *IEEE Micro*, 43(02), 76–85.

- [124] Sharma, H., Park, J., Mahajan, D., Amaro, E., Kim, J. K., Shao, C., Mishra, A., & Esmaeilzadeh, H. (2016). From high-level deep neural models to fpgas. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (pp. 1–12).
- [125] Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerry-Ryan, R. J., Saurous, R. A., Agiomyrgiannakis, Y., & Wu, Y. (2017). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 4779–4783).
- [126] Siemens EDA (accessed Nov 1, 2018). *Catapult High-Level Synthesis*.
- [127] Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- [128] Su, J.-W., Si, X., Chou, Y.-C., Chang, T.-W., Huang, W.-H., Tu, Y.-N., Liu, R., Lu, P.-J., Liu, T.-W., Wang, J.-H., Zhang, Z., Jiang, H., Huang, S., Lo, C.-C., Liu, R.-S., Hsieh, C.-C., Tang, K.-T., Sheu, S.-S., Li, S.-H., Lee, H.-Y., Chang, S.-C., Yu, S., & Chang, M.-F. (2020). 15.2 a 28nm 64kb inference-training two-way transpose multibit 6t sram compute-in-memory macro for ai edge chips. In *2020 IEEE International Solid-State Circuits Conference - (ISSCC)* (pp. 240–242).
- [129] Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V., Cui, X., Zhang, W., & Gopalakrishnan, K. (2019). *Hybrid 8-Bit Floating Point (HFP8) Training and Inference for Deep Neural Networks*. Curran Associates Inc.: Red Hook, NY, USA.
- [130] Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., & Zhou, D. (2020). Mobilebert: a compact task-agnostic bert for resource-limited devices. In *ACL*.
- [131] Tambe, T., Hooper, C., Pentecost, L., Jia, T., Yang, E.-Y., Donato, M., Sanh, V., Whatmough, P., Rush, A., Brooks, D., & Wei, G.-Y. (2021a). Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.
- [132] Tambe, T., Yang, E.-Y., Ko, G. G., Chai, Y., Hooper, C., Donato, M., Whatmough, P., Rush, A., Brooks, D., & Wei, G.-Y. (2021b). A 25mm<sup>2</sup> soc for iot devices with 18ms noise-robust speech-to-text latency via bayesian speech denoising and attention-based sequence-to-sequence dnn speech recognition in 16nm finfet. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*.
- [133] Tambe, T., Yang, E.-Y., Ko, G. G., Chai, Y., Hooper, C., Donato, M., Whatmough, P. N., Rush, A. M., Brooks, D., & Wei, G.-Y. (2023). A 16-nm soc for noise-robust speech and nlp edge ai inference with bayesian sound source separation and attention-based dnns. *IEEE Journal of Solid-State Circuits*, 58(2), 569–581.

- [134] Tambe, T., Yang, E. Y., Wan, Z., Deng, Y., Janapa Reddi, V., Rush, A., Brooks, D., & Wei, G.-Y. (2020). Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*.
- [135] Tambe, T., Zhang, J., Hooper, C., Jia, T., Whatmough, P., Zuckerman, J., Cassel, M., Loscalzo, E., Giri, D., Shepard, K., Carloni, L., Rush, A., Brooks, D., & Wei, G.-Y. (2023). A 12nm 18.1tflops/w sparse transformer processor with entropy-based early exit, mixed-precision predication and fine-grained power management. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*.
- [136] Tan, C., Tambe, T., Zhang, J. J., Fang, B., Geng, T., Wei, G.-Y., Brooks, D., Tumeo, A., Gopalakrishnan, G., & Li, A. (2022). Asap: Automatic synthesis of area-efficient and precision-aware cgras. In *Proceedings of the 36th ACM International Conference on Supercomputing (ICS)*.
- [137] Taylor, M. B. (2012). Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the 49th Annual Design Automation Conference, DAC '12* New York, NY, USA: Association for Computing Machinery.
- [138] Teerapittayanon, S., McDanel, B., & Kung, H. T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd International Conference on Pattern Recognition (ICPR)*, (pp. 2464–2469).
- [139] Arm Developer (2023). Cortex-a53. <https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53>.
- [140] Harvard Architecture, Circuits, and Compilers (2023). Flexasr: A reconfigurable hardware accelerator for attention-based seq-to-seq networks. <https://github.com/harvard-acc/FlexASR>.
- [141] Libeigen (2023). Eigen. <https://gitlab.com/libeigen/eigen>.
- [142] Project Ne10 (2023). Ne10 library. <https://projectne10.github.io/Ne10/>.
- [143] Toprak-Deniz, Z. et al. (2014). Distributed system of digitally controlled microregulators enabling per-core dvfs for the power8 tm microprocessor. In *ISSCC*.
- [144] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2020). Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*.
- [145] Tu, F., Wu, W., Yin, S., Liu, L., & Wei, S. (2018). Rana: Towards efficient neural acceleration with refresh-optimized embedded dram. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)* (pp. 340–352): IEEE.

- [146] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *ArXiv*, abs/1609.03499.
- [147] Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *ArXiv*, abs/1706.03762.
- [148] Venkataramani, S., Ranjan, A., Banerjee, S., Das, D., Avancha, S., Jagannathan, A., Durg, A., Nagaraj, D., Kaul, B., Dubey, P., & Raghunathan, A. (2017). Scaleddeep: A scalable compute architecture for learning and evaluating deep networks. *SIGARCH Comput. Archit. News*.
- [149] Venkataramani, S., Srinivasan, V., Wang, W., Sen, S., Zhang, J., Agrawal, A., Kar, M., Jain, S., Mannari, A., Tran, H., Li, Y., Ogawa, E., Ishizaki, K., Inoue, H., Schaal, M., Serrano, M., Choi, J., Sun, X., Wang, N., Chen, C.-Y., Allain, A., Bonano, J., Cao, N., Casatuta, R., Cohen, M., Fleischer, B., Guillorn, M., Haynie, H., Jung, J., Kang, M., Kim, K.-h., Koswatta, S., Lee, S., Lutz, M., Mueller, S., Oh, J., Ranjan, A., Ren, Z., Rider, S., Schelm, K., Scheuermann, M., Silberman, J., Yang, J., Zalani, V., Zhang, X., Zhou, C., Ziegler, M., Shah, V., Ohara, M., Lu, P.-F., Curran, B., Shukla, S., Chang, L., & Gopalakrishnan, K. (2021). Rapid: Ai accelerator for ultra-low precision training and inference. In *Proceedings of the 48th Annual International Symposium on Computer Architecture, ISCA '21* (pp. 153–166): IEEE Press.
- [150] Venkatesan, R., Shao, Y. S., Wang, M., Clemons, J., Dai, S., Fojtik, M., Keller, B., Klinefelter, A., Pinckney, N., Raina, P., Zhang, Y., Zimmer, B., Dally, W. J., Emer, J., Keckler, S. W., & Khailany, B. (2019). Magnet: A modular accelerator generator for neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (pp. 1–8).
- [151] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.
- [152] Wang, H., Zhang, Z., & Han, S. (2021). Spatten: Efficient sparse attention architecture with cascade token and head pruning. *2021 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.
- [153] Wang, J., Wang, X., Eckert, C., Subramaniyan, A., Das, R., Blaauw, D., & Sylvester, D. (2019). 14.2 a compute sram with bit-serial integer/floating-point operations for programmable in-memory vector acceleration. In *2019 IEEE International Solid-State Circuits Conference - (ISSCC)* (pp. 224–226).
- [154] Wang, Y., Qin, Y., Deng, D., Wei, J., Zhou, Y., Fan, Y., Chen, T., Sun, H., Liu, L., Wei, S., & Yin, S. (2023). An energy-efficient transformer processor exploiting dynamic weak relevances in global attention. *IEEE Journal of Solid-State Circuits*, 58(1), 227–242.

- [155] Wang, Z., Li, Z., Xu, L., Dong, Q., Su, C.-I., Chu, W.-T., Tsou, G., Chih, Y.-D., Chang, T.-Y. J., Sylvester, D., Kim, H. S., & Blaauw, D. (2020). An all-weights-on-chip dnn accelerator in 22nm ull featuring 24×1 mb erram. In *2020 IEEE Symposium on VLSI Circuits* (pp. 1–2).
- [156] Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. H. (2016). Learning structured sparsity in deep neural networks. *ArXiv*, abs/1608.03665.
- [157] Whatmough, P. N., Donato, M., Ko, G. G., Lee, S. K., Brooks, D., & Wei, G.-Y. (2020). CHIPKIT: An Agile, Reusable Open-Source Framework for Rapid Test Chip Development. *IEEE Micro*, 40(4), 32–40.
- [158] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J. R., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G. S., Hughes, M., & Dean, J. (2016). Google’s neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.
- [159] Xin, J., Tang, R., Lee, J., Yu, Y., & Lin, J. (2020). Deebert: Dynamic early exiting for accelerating bert inference. *ArXiv*, abs/2004.12993.
- [160] Ye, J., Lu, X., Lin, Z. L., & Wang, J. Z. (2018). Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *ArXiv*, abs/1802.00124.
- [161] Zadeh, A. H. & Moshovos, A. (2020). Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *53rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*.
- [162] Zhang, S. Q., McDanel, B., & Kung, H. (2022a). Fast: Dnn training under variable precision block floating point with stochastic rounding. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (pp. 846–860): IEEE.
- [163] Zhang, S. Q., McDanel, B., & Kung, H. T. (2022b). Fast: Dnn training under variable precision block floating point with stochastic rounding. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (pp. 846–860): IEEE Computer Society.
- [164] Zhang, S. Q., McDanel, B., Kung, H. T., & Dong, X. (2021). Training for multi-resolution inference using reusable quantization terms. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’21* (pp. 845–860). New York, NY, USA: Association for Computing Machinery.
- [165] Zhang, S. Q., Tambe, T., Cuevas, N., Wei, G.-Y., & Brooks, D. (2023). Camel: Co-designing ai models and embedded drams for efficient on-device learning. *arXiv*, abs/2305.03148.

- [166] Zhao, J., Dai, S., Venkatesan, R., Liu, M.-Y., Khailany, B., Dally, B., & Anandkumar, A. (2021). Lns-madam: Low-precision training in logarithmic number system using multiplicative weight update. *IEEE Transactions on Computers*, 71, 3179–3190.
- [167] Zhao, R., Hu, Y., Dotzel, J., Sa, C. D., & Zhang, Z. (2019). Improving neural network quantization without retraining using outlier channel splitting. *ArXiv*, abs/1901.09504.
- [168] Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., & Li, H. (2021). Learning n: M fine-grained structured sparse neural networks from scratch. *ArXiv*, abs/2102.04010.
- [169] Zhou, M., Duan, N., Liu, S., & Shum, H.-Y. (2020). Progress in neural nlp: Modeling, learning, and reasoning. *Engineering*, 6(3), 275–290.
- [170] Zhu, C., Han, S., Mao, H., & Dally, W. J. (2016). Trained ternary quantization. *ArXiv*, abs/1612.01064.
- [171] Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., & Zhu, J.-H. (2018). Discrimination-aware channel pruning for deep neural networks. In *Neural Information Processing Systems*.
- [172] Zimmer, B., Venkatesan, R., Shao, Y. S., Clemons, J., Fojtik, M., Jiang, N., Keller, B., Klinefelter, A., Pinckney, N., Raina, P., Tell, S. G., Zhang, Y., Dally, W. J., Emer, J. S., Gray, C. T., Keckler, S. W., & Khailany, B. (2020). A 0.32–128 tops, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm. *IEEE Journal of Solid-State Circuits*, 55(4), 920–932.