Diss. ETH No. 30368

Hardware-Software Co-Design for Energy-Efficient Neural Network Inference at the Extreme Edge

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

MORITZ LEO RUDOLF SCHERER
MSc ETH EEIT, ETH Zurich
born on 01.08.1996

accepted on the recommendation of

Prof. Dr. Luca Benini, examiner
Prof. Dr. Maurizio Martina, co-examiner

2024

Acknowledgments

There are many people I have to thank looking back on the past four years leading up to this dissertation.

First of all, my advisor Luca Benini, who has supported me throughout this journey and helped bring shape to novel ideas and refine my efforts; your guidance has inspired me to keep improving and challenge preconceived notions. In the same spirit, I would like to thank my first mentor Michele Magno, who took a chance on the *SmartAid* project Kiran, Paula, and I pitched to him in 2018, and who has supported me ever since.

Thanks go to the entire IIS staff. Frank, whose unparalleled technical expertise has left a lasting impression on me. The DZ team, Alfonso, Beat, and Zerun: your aid throughout my journey has been vital. Hansjörg, who has helped me from my very early days at IIS, and whose ingenuity in debugging electronics has saved me countless hours. Christoph, Adam, and Mateo, who keep the infrastructure running, maintaining the foundation for the research we do.

I would like to dedicate thanks to my 'metamates', Barbara, Chiao, Jorge, Warren, Reid, Ziyun, Syed, Kleber, Sai, Zhao, Korina, Andrew, Matt, Leslie, Lyle, and Dimitri, who have welcomed me as a research intern for half a year, helped me overcome technical challenges, and ensured I felt welcome throughout my stay. I will never forget this unique experience. Special thanks go to Jorge, who enabled this internship, and took care of me throughout it.

Next, I would like to acknowledge my colleagues. My original ETZ officemates, Florian, whose Swabian flair brought light to mundane work days, Samuel and Matheus, Maxim, and Andreas; the challenging pandemic years would not have been the same without you. My new officemates, Philip, Jannis, Viviane, and Lorenzo, and OAT-mates, Tim, Alfio, Marco, Yichao, Luca, Thorir, and Matteo who helped make the office feel alive, and with whom I have spent many working hours in the coffee corner.

Xia, Nils, and Michael whom I shared many meals with in our very own WR, the weekly restaurant; I cannot imagine a better group of people to share a hot pot.

Georg, whose unique sense of humor has entertained me throughout the years, and whose analytical mind has propelled our collaborations to technical maturity, starting from my own Master's thesis.

Manuel and Arpan, who worked tirelessly on the *Siracusa* project with me even when circumstances were suboptimal; your efforts made dealing with the challenges much easier.

Luka and Victor, who have helped grow *Deeploy*, not only through great technical contributions but also by being excellent teammates. Francesco, whose patience with me is unmatched: thanks for your advice, paper reviews, and frequent off-topic conversation throughout the years.

To Alfio, whose technical leadership in the *Kraken* tapeout has been foundational for large parts of this thesis: I have learned more from working with you than any book could teach, and you have been a valued friend on this journey.

Last but not least, thanks go to my family, who have supported me throughout my studies, and my friends Tom, Noelle, Michael, and Aitana who stuck with me through the hard times; thank you.

Abstract

Since the breakthrough success of AlexNet in the ILSVRC image recognition challenge in 2012, Deep Neural Networks (DNNs), and in particular Convolutional Neural Networks (CNNs), have become the standard algorithms for a wide range of data processing applications, including image processing, biomedical applications, Natural Language Processing (NLP) and many others. Advances in hardware technology, neural network architectures, and the increasing availability of training data have led to the rapid growth of DNN model sizes and, in turn, complexity.

While many recent Machine Learning (ML) algorithms require mastodontic computing capabilities and are increasingly run on datacenter-scale remote servers, small-scale DNNs have proven themselves helpful in processing highly domain-specific sensor data, leading to the emergence of the TinyML paradigm.

Under the TinyML paradigm, ML algorithms are deployed on resource-constrained embedded devices like Microntrollers (MCUs), which are typically integrated closely with sensors, such that data can be processed locally, without communication with the outside world. The goal of TinyML platforms is to process highly specialized DNN inference under real-time constraints while consuming power in the order of a few milliwatts and working with on-chip memory of at most a few megabytes. While such TinyML systems can achieve orders of magnitude of improvements in terms of energy efficiency and processing latency, the spartan compute capabilities and severely limited on-chip memory of MCU-class devices require

careful co-optimization of ML algorithms, computer architecture, and low-level deployment software.

In this thesis, we follow a holistic approach to designing highly energy-efficient TinyML systems. First, we present novel approaches optimize DNNs for TinyML applications, focusing on minimizing memory and compute requirements while retaining high accuracy. Next, we implement hardware accelerators for energy-efficient DNNs inference. Finally, we study the problem of automatic deployment of large-scale, complex DNNs, and design software tools to generate low-level performance- optimized code for heterogeneous System-on-chips (SoCs).

The first part of this thesis focuses on applications and quantization algorithms for ML-based time-series processing on MCUs. We study several sensor-driven TinyML applications, ranging from gesture recognition to keyword spotting. We develop broadly applicable techniques to adapt state-of-the-art ML algorithms for energy-efficient real-time inference in severely compute- and memory-constrained embedded devices.

Starting from circuit-level computer arithmetic optimizations, this thesis's second part focuses on designing a highly energy-efficient yet flexible Ternary Neural Network (TNN) accelerator called CUTIE. We complement this accelerator with a study of ternary DNN quantization algorithms and propose quantization strategies that optimally leverage CUTIE's architectural features, achieving the highest reported energy-efficiency of any fully digital TNN accelerator reported in literature.

We further present Temporal Convolutional Network (TCN) extensions for CUTIE, which enable the processing of time-series sensor data with the accelerator and the integration of the design into a RISC-V SoC in 22 nm FDX technology, called Kraken. We benchmark the deployment of a TCN for Dynamic Vision Sensor (DVS) gesture recognition on the produced Application-specific Integrated Circuit (ASIC), achieving the lowest per-gesture inference energy reported in literature.

In the final part, we move from algorithmic optimizations for TinyML applications to automatic DNN deployment code generation for heterogeneous SoCs. We design and implement a DNN deployment tool, Deeploy, which calculates optimal tiling strategies for computing systems using software-managed memories and generates optimized platform-specific code to execute modern large-scale DNNs on memory-constrained embedded SoCs.

Zusammenfassung

Seit dem durchschlagenden Erfolg von AlexNet bei dem ILSVRC-Wettbewerb im Jahr 2012 sind Deep Neural Networks (DNNs), insbesondere Convolutional Neural Networks (CNNs), zu den Standardalgorithmen für eine breite Palette von Datenverarbeitungsanwendungen geworden, einschließlich Bildverarbeitung, biomedizinischen Anwendungen, Natural Language Processing (NLP) und vielen anderen. Fortschritte in der Hardwaretechnologie, der Architektur neuronaler Netze und die zunehmende Verfügbarkeit von Trainingsdaten haben zu einem schnellen Wachstum der Grösse und somit der Komplexität von DNN-Modellen geführt.

Während viele neuere Machine Learning (ML)-Algorithmen mastodontische Rechenkapazitäten erfordern und zunehmend auf Grossrechnern verarbeitet werden, haben sich kleine DNNs bei der Verarbeitung anwendungsspezifischer Sensordaten als nützlich erwiesen, was zum Aufkommen des TinyML-Paradigmas geführt hat.

Unter dem TinyML-Paradigma werden ML-Algorithmen auf ressourcenbeschränkten eingebetteten Systemen wie Microntrollers (MCUs) eingesetzt, die typischerweise eng mit Sensoren gekoppelt sind, sodass Daten lokal verarbeitet werden können, ohne mit der Außenwelt zu kommunizieren. Das Ziel von TinyML-Plattformen ist es, hochspezialisierte DNN-Inferenz unter Echtzeitbedingungen zu ermöglichen, während sie nur wenige Milliwatt an Leistung verbrauchen und mit einem On-Chip-Arbeitsspeicher von höchstens einigen Megabytes auskommen. Obwohl solche TinyML-Systeme Grössenordnungen an Verbesserungen hinsichtlich Energieeffizienz und Verarbeitungslatenz er-

reichen können, erfordern die spartanischen Rechenkapazitäten und der stark begrenzte On-Chip-Speicher von MCUs eine sorgfältige Co-optimierung von ML-Algorithmen, Computerarchitektur und Compilern.

In dieser Dissertation verfolgen wir einen ganzheitlichen Ansatz zur Gestaltung energieeffizienter TinyML-Systeme. Zuerst stellen wir neuartige Ansätze vor, um DNNs für TinyML-Anwendungen zu optimieren, wobei wir uns auf die Minimierung von Speicher- und Rechenanforderungen konzentrieren, während wir eine hohe Genauigkeit beibehalten. Anschließend implementieren wir Hardwarebeschleuniger für energieeffiziente DNNs-Inferenz. Abschließend untersuchen wir das Problem der automatischen Bereitstellung von grossangelegten, komplexen DNNs und entwerfen Softwaretools zur Generierung von leistungsoptimiertem Code für heterogene System-on-chips (SoCs).

Der erste Teil dieser Dissertation konzentriert sich auf Anwendungen und Quantisierungsalgorithmen für ML-basierte Verarbeitung von Zeitreihen auf MCUs. Wir untersuchen mehrere sensorgetriebene TinyML-Anwendungen, von Gestenerkennung bis hin zu Keyword-Erkennung. Wir entwickeln breit anwendbare Methoden, um moderne ML-Algorithmen für energieeffiziente Echtzeit-Inferenz in stark rechen- und speicherbeschränkten Embedded-Geräten anzupassen.

Ausgehend von Computerarithmetikoptimierungen auf der Schaltungsebene konzentriert sich der zweite Teil dieser Dissertation auf die Entwicklung eines hochgradig energieeffizienten, aber dennoch flexiblen Ternary Neural Network (TNN)-Hardwarebeschleunigers namens CUTIE. Wir ergänzen diesen Hardwarebeschleuniger durch eine Studie über ternäre DNN-Quantisierungsalgorithmen und schlagen Quantisierungsstrategien vor, die die architektonischen Merkmale von CUTIE optimal nutzen, wodurch die höchste berichtete Energieeffizienz eines vollständig digitalen TNN-Beschleunigers in der Literatur erreicht wird.

Wir präsentieren darüber hinaus Temporal Convolutional Network (TCN)-Erweiterungen für CUTIE, die es ermöglichen, Zeitreihen-Sensordaten mit dem Hardwarebeschleuniger zu verarbeiten und das Design in einem RISC-V SoC in 22 nm FDX-Technologie, genannt Kraken, zu integrieren. Wir messen die Implementation

eines TCN für Dynamic Vision Sensor (DVS)-Gestenerkennung auf dem produzierten Application-specific Integrated Circuit (ASIC) und erreichen dabei die niedrigste pro-Geste-Inferenzenergie, die bisher in der Literatur berichtet wurde.

Im letzten Teil wechseln wir von algorithmischen Optimierungen für TinyML-Anwendungen zur automatischen DNN-Codegenerierung für heterogene SoCs. Wir entwerfen und implementieren ein DNN-Codegenerierungstool, Deeploy, das optimale Strategien zur Aufteilung der Netzwerkberechnung für Rechensysteme mit softwareverwalteten Speichern berechnet und optimierten plattformspezifischen Code generiert, um moderne umfangreiche DNNs auf speicherbeschränkten Embedded-SoCs auszuführen.

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, Deep Neural Networks (DNNs), especially Convolutional Neural Networks (CNNs), have positioned themselves as the cornerstone algorithms across a multitude of data processing applications, spanning image processing, biomedical fields, Natural Language Processing (NLP), and many more. Significant advancements in hardware technology have propelled this surge in adoption, the evolution of neural network architectures, and the burgeoning availability of training data, contributing to the exponential growth in the size and complexity of DNN models.

The widespread integration of DNNs into mainstream technology has been primarily facilitated by breakthroughs in Graphics Processing Unit (GPU) technology. The seminal AlexNet, often regarded as the catalyst for the current explosion in Machine Learning (ML) research [1], demonstrated the potential of leveraging multiple General-Purpose Graphics Processing Units (GPGPUs) for neural network computation. This milestone has shifted the focus of DNN research towards harnessing the power of datacenter-class compute clusters to scale models up to billions of parameters.

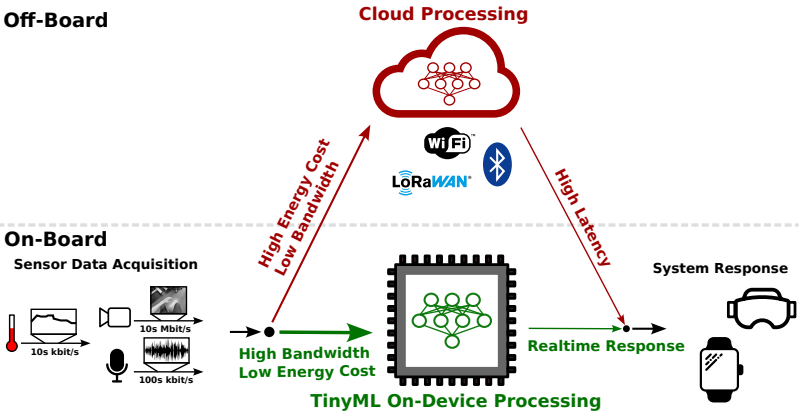

Figure 1.1: Comparison of cloud-based processing (red) and TinyML on-device processing (green); the TinyML approach minimizes energy consumption and latency for realtime, sensor-driven applications.

Simultaneously, research into network topologies has been complemented by algorithmic optimizations such as quantization and pruning. These optimizations, coupled with their demands for complex dataflow and non-standard computer arithmetic, have paved the way for developing novel hardware accelerators designed to effectively capitalize on these complexity-reduction techniques [2–4].

Adding to this, the slowdown in Moore's law [5] and the halt of Dennard scaling [6] emphasize the increasing need for specialized acceleration of DNN workloads. This need has become a critical research area and a practical requirement for further advancing DNN technologies. This trend is evident in the ongoing efforts of nearly all major tech companies, who are actively developing custom hardware accelerators for datacenter-scale processing of DNN workloads in cloud environments [2, 7].

In contrast to centralized deployment schemes, the emerging field of TinyML aims to empower the processing of neural networks at the local level directly on the device where data originates [3, 8]. Con-

sequently, TinyML's DNN inference faces much more stringent constraints, including limited hardware resources, stringent power consumption limitations, and the imperative for minimal processing latency. However, the promise of TinyML extends beyond these challenges, as it stands to become a pivotal enabler of groundbreaking applications far surpassing the capabilities of current systems. One compelling example is the evolution of Virtual Reality (VR) and Augmented Reality (AR) technologies [9], which demand substantial advancements in DNN inference energy efficiency and accuracy of concurrent real-time algorithms that transform gestures, speech, and user gaze into actionable information [9–11]. A comparison between the cloud-based processing approach and on-device processing is shown in Figure 1.1.

The challenges presented to the burgeoning field of TinyML motivate and demand innovation across the deep learning technology stack, including hardware accelerators, System-on-chip (SoC) architectures, DNN quantization & deployment software, and DNN architectures.

1.2 Machine Learning on the Extreme Edge: TinyML

TinyML represents the intersection of ML algorithms and ultra-low-power embedded systems, aiming to bring the advanced information extraction capabilities of DNNs to the very edge of the technology frontier onto the smallest and often battery-powered devices such as wearables, sensors, and various Internet of Things (IoT) devices [12]. The resource constraints of this class of devices require meticulous optimization to achieve tasks previously thought impractical for such challenging environments [13].

At its core, TinyML is about overcoming the limitations imposed by the minimal computational resources, limited memory in the order of kilo- or few megabytes, and stringent power constraints of Microntrollers (MCUs) and other low-power devices [3, 8]. It involves not only the downsizing of existing ML models without significant loss in accuracy [10, 14] but also the development of new algorithms and

techniques specifically designed for this scale [15]. Combining these approaches enables the deployment of intelligent functions directly on MCU-class devices, reducing the need for constant internet connectivity and thus addressing concerns related to data privacy, security, and latency.

Some key applications of TinyML include the processing of sensor data like audio, low-resolution images, and accelerometer data close to their source; as such, DNNs for TinyML applications can be much more optimized for their specific sensors and deployment environments than their datacenter-scale counterparts. Translating these constrained deployment scenarios into lower DNN complexity is paramount to meeting the hardware constraints of TinyML systems, as smaller models directly translate to lower inference latency and higher energy efficiency, enabling longer battery life on sensor nodes.

Besides DNN architecture optimization, one of the essential design methods for TinyML is quantization, which aggressively reduces the memory footprint of DNN models by representing operands and intermediate results using fewer bits. This translates to a significantly reduced memory footprint of DNNs, which addresses the memory constraint.

Quantization for TinyML systems

Neural Network (NN) models have proven themselves highly resilient to errors introduced by approximate computing techniques where intermediate results and network parameters are rounded to fit within a single byte without any retraining in a process called Post-Training Quantization (PTQ) [15, 16]. The compression of network parameters and activations to 8 bit significantly reduces their memory load compared to their single floating point precision equivalents.

However, quantizing DNNs to perform inference with sub-byte operands requires a more sophisticated approach than PTQ, as it often leads to significant degradation in model accuracy if not carefully managed [15, 17]. Techniques such as Quantization-Aware Training (QAT) come into play here, where the quantization process is integrated into the training phase itself [16, 18]. This method

allows the model to learn the quantization noise during training, making it more robust to the precision loss. Additionally, advanced techniques like mixed precision training [19], where different parts of the model use different quantization levels. Custom quantization schemes tailored to the specific characteristics of the model and its application domain can further mitigate the accuracy loss.

Dedicated accelerators can leverage more aggressive sub-byte quantization to achieve massive improvements in energy efficiency and throughput [20] due to a significant reduction in circuit complexity. For example, while a 32 bit integer product requires a large number of hardware adders to be performed in a single cycle, a binary product only requires a single XOR gate [21].

RISC-V SoCs for TinyML

With the introduction of the RISC-V Instruction Set Architecture (ISA) [22] and the open-source hardware movement, the playing field for exploring trade-offs in computer architecture, especially for MCU-class devices, has rapidly increased [23]. The RISC-V ISA, is designed with support for extensions in mind. This design space has been explored in recent years to develop application-optimized yet general-purpose cores. Notably, the xPULP ISA extension family adds support for low-cost hardware optimizations for Digital Signal Processing (DSP) and ML applications, like integer Single Instruction Multiple Data (SIMD) support, hardware loops, and post-load increment instructions [20].

However, while ISA extensions help to reduce the number of instructions required per operation, their potential is ultimately limited by Flynn's bottleneck [24], referring to the fact that every executed instruction must be fetched and decoded, which forms a ceiling on the achievable number of operations per cycle and achievable energy efficiency in processor architectures. Accelerators have found their way into most state-of-the-art TinyML systems as a natural extension of core-based processing [12] to address this fundamental constraint of all common computers.

Hardware Accelerators for TinyML

Hardware accelerators for tinyML applications present a unique set of challenges and considerations, ranging from the design of their data path to the integration within an SoC. However, overcoming these challenges can afford significant improvements in throughput and energy efficiency over conventional core-based processing and allows the exploration of algorithms that might be unfeasible or inefficient on general-purpose computers [25].

A fundamental trade-off in accelerator design is flexibility versus efficiency; generally, the more specialized the accelerator is for a particular task or algorithm, the higher its efficiency, but the lower its flexibility. This means that while a highly specialized accelerator can perform specific operations with exceptional speed and low power consumption, it may not be adaptable to different algorithms or tasks without significant redesign, rendering it obsolete. Conversely, a more flexible accelerator, capable of handling various tasks, may not achieve the same level of operational efficiency for any single task compared to its specialized counterpart. This trade-off is critical in the context of tinyML, where the specific requirements of an application must be carefully balanced against the inherent resource constraints of extreme edge devices.

This trade-off applies to virtually all design aspects of accelerators, but the most significant impact is usually found on their lowest level, where ML accelerators mainly compute matrix products [4,9,26]. The design space for datapath architectures implementing matrix multiplications is vast; it encompasses a wide range of architectural choices, from the arrangement of processing elements to data storage and retrieval methods. These choices significantly impact the accelerator's performance, energy efficiency, and silicon area usage. For instance, some designs opt for a systolic array architecture that efficiently handles data flow for matrix operations, enhancing parallelism and reducing memory access latency at significant energy cost spent on data movement through the array [27]. Other approaches to optimizing datapath efficiency include In-Memory Computing (IMC), where operations are performed directly in on-chip Static Random Access Memory (SRAM) macros rather than digital compute units.

Additionally, the choice between on-chip versus off-chip memory, the strategy for data reuse and caching, and the precision of computation (e.g., floating-point versus fixed-point arithmetic) further diversify the design space.

To make full use of the optimization opportunities of deeply specialized hardware accelerators while maintaining support for novel DNN architectures, relying on a single accelerator or compute engine is often not sufficient; integrating multiple accelerators on a single SoC is an approach that aims to overcome this issue. However, heterogeneity in SoC designs introduces further challenges for TinyML.

Heterogeneous SoCs for TinyML

Traditional MCUs are not designed for high-performance computing, focusing instead on rich peripherals and simple, low-power cores that allow them to interact with their environment. In contrast, the tinyML paradigm encourages the design of High-Performance Computing (HPC) inspired systems, favoring multi-core compute clusters [28, 29] and dedicated hardware accelerators [25, 26, 30] over single core systems. This approach centers around the utilization of multi-core processors for general computational tasks, complemented by dedicated hardware accelerators specifically designed for Artificial Intelligence (AI) and machine learning workloads, such as Neural Processing Units (NPUs) in a single Application-specific Integrated Circuit (ASIC).

Leveraging synergies between multi-core clusters and hardware accelerators enables highly efficient processing of DNNs. Multi-core compute clusters can handle various tasks with parallel processing capabilities. At the same time, hardware accelerators are optimized for high-speed, low-power execution of specific computations, such as matrix multiplications or convolutions in various quantization schemes.

Adopting such a dual approach mitigates the limitations of relying solely on general-purpose computing or specialized acceleration, offering a versatile platform that can adeptly meet the computational demands of advanced AI algorithms. It aligns well with the constraints of extreme edge computing, where efficient use of power is

paramount. However, optimizing DNNs for such heterogeneous computing architectures requires sophisticated software tools, as the absence of Memory-Management Units (MMUs) and Operating Systems (OSs) in MCUs requires software solutions that are capable of splitting workloads into chunks that fit within the tight memory constraints of MCUs [3].

NN Deployment on Heterogeneous SoCs

While DNNs have traditionally been trained and executed on GPGPUs, where compute and memory resources can be scaled arbitrarily, and MMUs and OSs take over low-level hardware management tasks, model deployment for TinyML requires an entirely different approach, as these management tasks must be handled in software.

To exploit the compute engines of heterogeneous SoCs, deployment tools must not only generate code for executing optimized low-level compute kernels but also orchestrate overlapping memory transfers between various hierarchy levels [3] and calculate an execution schedule to achieve low data marshaling overheads and high compute utilization.

Overcoming these challenges is paramount to enabling efficient model execution on baremetal hardware. However, to manage state-of-the-art networks, deployment tools must also account for the memory constraints of targeted devices; this is especially challenging, as modern edge AI MCUs feature multiple memory levels [3, 31], requiring operator tiling, which must account for geometrical and performance constraints. Generating code for such platforms typically requires a static memory allocation strategy, which implements the tiling strategy to guarantee correct inference on the device.

This thesis studies the TinyML technology stack holistically, addressing end-to-end energy efficiency challenges from the modeling, quantization, deployment, and acceleration angles.

1.3 Outline

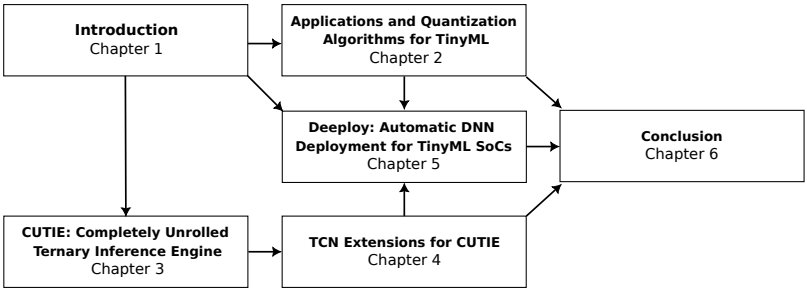

Figure 1.2: Block diagram of this thesis' structure

In the following, we outline this thesis, as shown in Figure 1.2. The content presented throughout this thesis has already been peer-reviewed and published in conference proceedings and journal papers.

Chapter 2

The second chapter introduces two different TinyML applications for extracting high-level information from noisy sensor data, gesture recognition, and keyword spotting. It discusses approaches based on Temporal Convolutional Network (TCN) and Transformer networks. Both applications implement 8 bit quantization, and deployment flows for extreme edge devices. The presented implementations are measured on their target platforms, achieving state-of-the-art energy efficiency and accuracy for their respective tasks.

Chapter 3

This chapter introduces the CUTIE accelerator and discusses its key design characteristics. The accelerators are synthesized and implemented in a prototype backend flow in GlobalFoundries 22 nm technology, which is used to gather post-layout power simulation estimates.

It further demonstrates the impact of quantization and sparsity on CUTIE's energy efficiency.

Chapter 4

The fourth chapter introduces TCN extensions for CUTIE, introduced in Chapter 3, to enable support for networks similar to those described in Chapter 2. Furthermore, the silicon implementation of the extended accelerator within the *Kraken* SoC in GlobalFoundries 22 nm technology is discussed, and end-to-end measurements on the fabricated ASIC are presented.

Chapter 5

The fifth chapter introduces Deeploy, a bottom-up compiler for heterogeneous TinyML SoCs, like the SoC presented in Chapter 4, to enable state-of-the-art TinyML DNNs like the ones introduced in Chapter 2. Through the deployment of a complex decoder-only Transformer network on the *Siracusa* SoC, Deeploy's flexibility and adaptability are demonstrated, leading to leading-edge energy efficiency and throughput of a Small Language Model (SLM) on an extreme edge device.

Chapter 6

Chapter six summarizes this thesis and offers an outlook on future challenges and opportunities for TinyML devices and algorithms.

1.4 Contributions

The main contributions of this thesis, as well as the related publications, are summarized as follows:

1. The training of a novel mixed CNN and TCN model for gesture recognition, its quantization, and deployment on a state-of-the-art RISC-V multi-core TinyML system and measurements of the power consumption on the device (Chapter 2.1, [32]).
2. The design, quantization, and deployment of a linear transformer model for audio recognition on an off-the-shelf ARM

microcontroller, achieving above state-of-the-art word recognition accuracy and real-time inference on the 96 MHz MCU. On-device power measurements complete the study (Chapter 2.2, [33]).

3. The design of an ultra-low energy-per-inference digital accelerator for Ternary Neural Networks (TNNs), CUTIE, as well as an exploration of quantization schemes for TNNs and their impact on adder tree activity in the accelerator. It is demonstrated that the accelerator's design approach of maximizing architecture parallelism and focusing on ternary over binary values optimizes energy efficiency. Furthermore, post-layout silicon estimates of the accelerator are provided (Chapter 3, [34]).

4. The design and implementation of TCN extensions for the CUTIE accelerator, as well as a silicon implementation of the design in GlobalFoundries 22 nm technology. Silicon measurements of the fabricated device improve on the post-layout estimates, achieving 1 POp/J in a wholly digital design (Chapter 4, [35]).

5. The design and implementation of a novel compiler, Deeploy, for DNNs on TinyML devices. Deeploy solves the tiling and static memory allocation problems for large neural networks deployed on MCUs with multiple memory hierarchy levels in a single constraint program and generates low-level C code supporting the integration of user-provided kernels. We also present an in-depth study on end-to-end performance and power consumption for autoregressive transformer inference on a heterogeneous multi-core system (Chapter 5, [36]).

1.5 List of Publications

Most of the material covered in this thesis has been published in the following conference and journal papers:

[32] M. Scherer, M. Magno, J. Erb, P. Mayer, M. Eggimann, and L. Benini, "TinyRadarNN: Combining Spatial and Temporal Convolutional Neural Networks for Embedded Gesture Recognition With Short Range Radars,"

IEEE Internet of Things Journal, vol. 8, no. 13, pp. 10 336–10 346, Jul. 2021

[33] M. Scherer, C. Cioflan, M. Magno, and L. Benini, "Work In Progress: Linear Transformers for TinyML," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2024

[34] M. Scherer, G. Rutishauser, L. Cavigelli, and L. Benini, "CUTIE: Beyond PetaOp/s/W Ternary DNN Inference Acceleration With Better-Than-Binary Energy Efficiency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1020–1033, Apr. 2022

[35] M. Scherer, A. D. Mauro, T. Fischer, G. Rutishauser, and L. Benini, "TCN-CUTIE: A 1,036-TOp/s/W, 2.72-µJ/Inference, 12.2-mW All-Digital Ternary Accelerator in 22-nm FDX Technology," *IEEE Micro*, vol. 43, no. 1, pp. 42–48, Jan. 2023

[36] M. Scherer, L. Macan, V. Jung, P. Wiese, A. Burrello, F. Conti, and L. Benini, "Deeploy: Enabling Energy-Efficient Deployment of Small Language Models On Heterogeneous Microcontrollers," Mar. 2024, under Review at IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)

Further publications by the author that are not or only partially covered in this thesis:

[37] M. Scherer, K. Menachery, and M. Magno, "SmartAid: A Low-Power Smart Hearing Aid For Stutterers," in *2019 IEEE Sensors Applications Symposium (SAS)*, Mar. 2019, pp. 1–6

[38] A. Di Mauro, M. Scherer, J. F. Mas, B. Bougenot, M. Magno, and L. Benini, "FlyDVS: An Event-Driven Wireless Ultra-Low Power Visual Sensor Node," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Feb. 2021, pp. 1851–1854

[39] M. Scherer, P. Mayer, A. di Mauro, M. Magno, and L. Benini, "Towards Always-on Event-based Cameras for Long-lasting Battery-operated Smart Sensor Nodes," in *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, May 2021, pp. 1–6

[40] A. Burrello, M. Scherer, M. Zanghieri, F. Conti, and L. Benini, "A Microcontroller is All You Need: Enabling Transformer Execution on Low-Power IoT Endnodes," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, Aug. 2021, pp. 1–6

[41] A. Burrello, F. B. Morghet, M. Scherer, S. Benatti, L. Benini, E. Macii, M. Poncino, and D. J. Pagliari, "Bioformers: Embedding Transformers for Ultra-Low Power sEMG-based Gesture Recognition," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2022, pp. 1443–1448

[**42**] A. Di Mauro, M. Scherer, D. Rossi, and L. Benini, "Kraken: A Direct Event/Frame-Based Multi-sensor Fusion SoC for Ultra-Efficient Visual Processing in Nano-UAVs," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, Aug. 2022, pp. 1–19

[**43**] M. Scherer, A. Di Mauro, G. Rutishauser, T. Fischer, and L. Benini, "A 1036 TOp/s/W, 12.2 mW, 2.72 μJ/Inference All Digital TNN Accelerator in 22 nm FDX Technology for TinyML Applications," in *2022 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Apr. 2022, pp. 1–3

[**44**] M. Scherer, F. Sidler, M. Rogenmoser, M. Magno, and L. Benini, "WideVision: A Low-Power, Multi-Protocol Wireless Vision Platform for Distributed Surveillance," in *2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2022, pp. 394–399

[**45**] P. Busia, A. Cossettini, T. M. Ingolfsson, S. Benatti, A. Burrello, M. Scherer, M. A. Scrugli, P. Meloni, and L. Benini, "EEGformer: Transformer-Based Epilepsy Detection on Raw EEG Traces for Low-Channel-Count Wearable Continuous Monitoring Devices," in *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Oct. 2022, pp. 640–644

[**46**] G. Rutishauser, M. Scherer, T. Fischer, and L. Benini, "Ternarized TCN for μJ/Inference Gesture Recognition from DVS Event Frames," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 736–741

[**47**] M. Scherer, M. Eggimann, A. D. Mauro, A. S. Prasad, F. Conti, D. Rossi, J. T. Gómez, Z. Li, S. S. Sarwar, Z. Wang *et al.*, "Siracusa: A Low-Power On-Sensor RISC-V SoC for Extended Reality Visual Processing in 16nm CMOS," in *ESSCIRC 2023- IEEE 49th European Solid State Circuits Conference (ESSCIRC)*, Sep. 2023, pp. 217–220

[**48**] G. Rutishauser, M. Scherer, T. Fischer, and L. Benini, "7 μJ/Inference End-to-End Gesture Recognition from Dynamic Vision Sensor Data Using Ternarized Hybrid Convolutional Neural Networks," *Future Generation Computer Systems*, vol. 149, pp. 717–731, Dec. 2023

[**49**] A. Mattei, M. Scherer, C. Cioflan, M. Magno, and L. Benini, "Securing Tiny Transformer-based Computer Vision Models: Evaluating Real-World Patch Attacks," in *9th World Forum on the Internet of Things (WF-IoT 2023)*, 2023

[**50**] G. Islamoglu, M. Scherer, G. Paulin, T. Fischer, V. J. Jung, A. Garofalo, and L. Benini, "ITA: An Energy-Efficient Attention and Softmax Accelerator for Quantized Transformers," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2023, pp. 1–6

[**30**] A. S. Prasad, M. Scherer, F. Conti, D. Rossi, A. Di Mauro, M. Eggimann, J. T. Gómez, Z. Li, S. S. Sarwar, Z. Wang *et al.*, "Siracusa: A 16 nm Heterogenous RISC-V SoC for Extended Reality with At-MRAM Neural Engine," *ArXiv*, no. arXiv:2312.14750, Dec. 2023, accepted for publication at IEEE Journal of Solid-State Circuits

[**51**] P. Busia, A. Cossettini, T. M. Ingolfsson, S. Benatti, A. Burrello, V. J. B. Jung, M. Scherer, M. A. Scrugli, A. Bernini, P. Ducouret *et al.*, "Reducing False Alarms in Wearable Seizure Detection with EEGformer: A Compact Transformer Model for MCUs," *IEEE Transactions on Biomedical Circuits and Systems*, pp. 1–13, 2024

[**52**] V. J. B. Jung, A. Burrello, M. Scherer, F. Conti, and L. Benini, "Optimizing the Deployment of Tiny Transformers on Low-Power MCUs," *ArXiv*, no. arXiv:2404.02945, Apr. 2024, manuscript submitted for review at IEEE Transactions on Computers

[**53**] V. Potocnik, A. D. Mauro, C. Leitner, M. Scherer, G. Rutishauser, L. Lamberti, and L. Benini, "Kraken: An Open-Source RISC-V SoC for Ultra-Low Power Multi-Modal Perception," *ArXiv*, Apr. 2024

Chapter 2

Applications and Quantization Algorithms for TinyML

In this chapter, we focus on sensor-driven, time series classification TinyML applications and methodologies for DNN size reduction, quantization, and deployment on extreme edge devices, achieving state-of-the-art accuracy, throughput, and energy efficiency.

The problem of time-series classification is pervasive in TinyML applications, as most sensors used in embedded devices produce low-dimensional time series, like audio, radar, or accelerometer data. The approaches described in this chapter, TCNs and Transformers, are shown to be adaptable to the TinyML domain.

Section 2.1 introduces a dual-stage algorithm for classifying gestures from radar data consisting of a *per-frame* CNN and a *per-sequence* TCN. Using this dual approach of extracting spatial features from a CNN and temporal features from a TCN not only minimizes the model's memory footprint, compressing it to 92 kB, but outperforms larger, traditional autoregressive models like Long Short-Term Mem-

orys (LSTMs). We demonstrated end-to-end deployment and integration with the sensor on the GAP8 MCU, achieving real-time prediction in a power envelope of only 21 mW. We further collected and open-sourced a dataset collected for this work, which contains 11 gestures classes from 26 subjects.

Section 2.2 proposes a quantization and deployment methodology to enable the use of linear attention on off-the-shelf low-power low-cost 32-bit MCUs. As a use case of this methodology, we present the WaveFormer, a 130 KB neural network based on a linear attention architecture that achieves a new state-of-the-art accuracy of 99.45 % on the Google Speech 35 V2 dataset. We further demonstrate the 8-bit quantization and embedded deployment of the WaveFormer on the Apollo 4, an ARM Cortex-M4 platform, achieving a latency of 741 ms, adequate for real-time operation, as well as an energy consumption of just 8.7 mJ per inference.

Section 2.1 has been published in a slightly different form in IEEE Internet of Things Journal (IOTJ)[1] and contains minor modifications. Section 2.2 has been adapted from a publication at the 2024 Design, Automation & Test in Europe Conference & Exhibition Conference (DATE)[2], and contains further explanations on deployment results and quantization methodology.

[1] © 2021 IEEE. Reprinted, with permission, from M. Scherer, M. Magno, J. Erb, P. Mayer, M. Eggimann, and L. Benini, "TinyRadarNN: Combining Spatial and Temporal Convolutional Neural Networks for Embedded Gesture Recognition With Short Range Radars," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 336–10 346, Jul. 2021

[2] © 2024 IEEE. Reprinted, with permission, from M. Scherer, C. Cioflan, M. Magno, and L. Benini, "Work In Progress: Linear Transformers for TinyML," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2024

2.1 TinyRadarNN: Combining Spatial and Temporal CNNs

2.1.1 Introduction

Human-computer Interface (HCI) and Human activity recognition (HAR) systems provide a plethora of attractive application scenarios with a wide array of solutions, strategies, and technologies [54,55]. Hand gestures are one of the most natural ways for people to interact, control and engage with devices and machines in the IoT paradigm [56]. For this reason it is not surprising that one of the emerging technologies in the context of wearable devices is gesture recognition [57]. Traditionally, the approaches for capturing human gestures are based on image data or direct measurements of movement, i.e. by using motion sensors [57–59]. The main types of sensors used in literature are cameras with and without depth perception, force-sensitive resistors, capacitive elements and accelerometers to measure the movement of the subject directly [60]. While these approaches have been shown to work well in controlled settings, robustness remains a challenge in real-world application scenarios. Image-based approaches have to deal with well-known environmental challenges like subject occlusion and variability in brightness, contrast, exposure and other parameters [61]. Another drawback of image-based solutions is the comparatively high power consumption, with commercial sensors like the Kinect sensors having power consumptions in the order of watts [62]. Wearable systems using motion-based sensing are much less affected by environmental variability and typically use significantly less power, but are more difficult to adapt to differences in user physique and behaviour. Approaches based on Wi-Fi have also been studied. On single subjects they have been shown to achieve high accuracy [59, 63], but are generally restricted to coarse or full-body gestures, due to the low spatial resolution, signal strength and susceptibility to electromagnetic interference and multi-path reflections [64–66].

A very promising, novel sensing technology for hand gesture recognition is based on high frequency and short-range pulsed RADAR sensors [67]. RADAR technology can leverage the advantages of image-based recognition with reduced challenges from environmental vari-

ability. The electromagnetic RADAR waves can propagate through matter, such that it can potentially record responses even if placed behind clothing. Furthermore, recently proposed designs based on novel sensor implementations can fit within a low power budget [67], compatible with the constraints of wearable devices. However, achieving RADAR-based gesture recognition on the highly constrained computing platforms available on wearables remains an open challenge.

Battery-operated wearable devices for the Internet of Things, especially those used for machine learning and data mining applications, typically host an ARM Cortex-M or RISC-V based microcontroller, which can achieve power consumption in the order of a few milliwatts and computational speeds in the order of hundreds of MOp/s [68–71], while offering memory storage of at most a few megabytes. Fitting within these limited computational resources to run machine learning algorithms especially for high-bandwidth sensors, such as imagers or RADARs, remains challenging [72, 73]. Recently, several research efforts have started to focus on specialized hardware to run machine learning algorithms, and in particular neural networks on power-constrained devices [72, 74–76]. Parallel architectures leveraging near-threshold operation and multi-core clusters, enabling significant increases in energy efficiency, have been explored in recent years with different application workloads [77] and low-power systems [29].

The main state-of-the-art approaches to machine learning-based time-sequence modelling for gesture recognition are Hidden Markov Models (HMM) [78] and LSTM [79] networks, which both use an internal state to model the temporal evolution of the signal. In recent years especially, Artificial Neural Networkss (ANNs) have seen a rapid increase in popularity, with most recent works relying on LSTM-based approaches [80, 81]. The recently proposed RADAR sensing platform Soli, jointly developed by Infineon and Google, has been studied in different works, most prominently by Wang et al. [82]. They propose an LSTM model that achieves an accuracy of above 90% over 11 classes.

In contrast to the state-based modelling of the input signal, TCNs are stateless in the sense that their computation model does not depend on the input. This means that they can compute sequential outputs in

parallel, unlike LSTMs or HMMs [83]. Furthermore, since they only use stateless layers, TCNs use significantly less memory for buffering feature maps compared to LSTMs, defusing the memory bottleneck on embedded platforms. TCNs have increasingly been adopted in many application scenarios where the classification of data is heavily linked to its temporal properties, for example, biomedical data [84] or audio data [85].

This Subsection proposes a novel embedded, highly accurate temporal convolutional neural network architecture, optimized for low-power microcontrollers. The proposed model achieves both a memory footprint of less than 100 KB, as well as achieving a per-sequence inference accuracy of around 86.6% for 11 challenging gesture classes, trained on a multi-user dataset, and 92.4% for a single-user dataset. We exploit a novel, low-power short-range A1 RADAR sensors from Acconeer[3] to acquire two rich and diverse datasets, one for a single user and one for a total of 26 users, each containing 11 gestures. Further, we leverage a multi-core RISC-V based embedded processor taking advantage of the emerging parallel ultra-low power (PULP) computing paradigm to enable the execution of complex algorithmic flows on power-constrained devices. Similar to Soli, possible deployment scenarios for the algorithm and processing platform include smart devices like smartphones[4] and smart thermostats[5] and even wearables with small form factor like smartwatches and hearing aids [29]. Due to the small footprint of less than 30 mm^2, the RADAR sensor can be easily integrated into most wearable devices [86].

We show that highly-accurate, real-time hand gesture recognition within a power budget of around 120 mW, including the sensor and processing consumption, is possible with the proposed sensor and computing platform. Experimental evaluations with a working prototype demonstrate both the power consumption and the high accuracy and are presented in this Section.

The main contribution of this Section can be summarized as follows:

[3]https://www.acconeer.com/products
[4]https://ai.googleblog.com/2020/03/soli-radar-based-perception-and.html
[5]https://www.gearbrain.com/new-google-nest-hub-soli-2649744781.html

- Design and implementation of a TCN network architecture optimized for low-power hand gesture recognition on microcontrollers, achieving state-of-the-art accuracy with a total memory footprint of less than 512 KB.
- Acquistion and labeling of an open-source gesture recognition dataset featuring 11 challenging, fine-grained hand gestures recorded with the low-power Acconeer A1 pulsed RADAR sensor to provide a baseline dataset for future research.
- Implementation of the proposed model in a novel parallel RISC-V based microcontroller, featuring 8 specialized parallel cores for processing and 512 KB of on-chip memory. The novel, power-optimized architecture of the processors enables a full-system power consumption below 100 mW in full active mode.
- Evaluation of the benefits of the algorithm in terms of accuracy, energy efficiency and inference speed, showing that the processor consumes only 21 mW, which is orders of magnitude less power for real-time prediction compared to the state-of-the-art, at a comparable level of accuracy.

2.1.2 Related Work

Hand gesture recognition is a widely investigated field. However, it is difficult to put all the research into context, as there are many different categories of hand gestures, which vary in complexity. Also, depending on the number of modelled gestures, the sensor used, and how well diversified the studied dataset is, accuracies vary greatly. In this Subsection we review RADAR based approaches which are most directly comparable with our work. We refer the interested reader to [64,87–92] for image-based, inertial and RF-based gesture recognition.

RADAR-based gesture recognition

Some research has been conducted to exploit RADAR systems or radio signals to predict hand gestures. The approaches vary in terms of the application scenario, as well as accuracy and power efficiency. Different models without explicit sequence modelling have been employed in the past, a sample of which is discussed here. Kim et al. use pulsed

radio signals to determine static hand gestures by analysing the differences between reflected waveforms with the help of a 1D CNN. Accuracies of over 90% are achieved for American Sign Language (ASL) hand signs using a CNN and micro-Doppler signatures [93]. In their feasibility analysis, Kim and Toomajian use deep convolutional neural networks to classify ten hand-gestures using micro-doppler signatures from a pulsed RADAR. Their offline prediction algorithm reaches an accuracy of 85.6% on a single participant [94]. Using a similar approach based on micro-doppler signatures and a Frequency-Modulated Continuous Wave (FMCW) RADAR, Sun et al. showed that inference accuracy of over 90% on a nine gesture dataset recorded from a stationary RADAR for driving-related gestures is possible [95].

Different works have used combinations of LSTM cells or Hidden Markov Models combined with different pre-processing strategies and convolutional layers to classify both coarse- and fine-grained gestures with the help of time-sequence modelling. Hazra et al. present a FMCW-based system which is trained to recognize eight gestures, reaching an accuracy of over 94% [96]. Targeting embedded, low-power applications, Lien et al. developed a high-frequency short-range RADAR specifically for the purpose of hand-gesture recognition, called Soli. They implement a neural network to classify four hand gestures. Their final implementation uses a random forest classifier on those features with an optional bayesian filter of the random forest output. They use four micro-gestures, which they call "virtual button" (pinch index), "virtual slider" (sliding with index finger over thumb), "horizontal swipe" and "vertical swipe". On those four gestures, they achieve a per-sample accuracy of 78.22% and a per-sequence accuracy of 92.10% for the bayesian filtered random forest output [67]. Choi et al. used the Soli sensor and a self-recorded 10 gesture dataset featuring ten participants to train an LSTM-based neural network. They achieve an accuracy of over 98% using a GPU for inference computation [97]. Using the Soli sensor, Wang et al. propose a machine learning model to infer the hand motions contained in the RADAR signal, based on an ANN network containing both convolutional layers and LSTM cells. They employ a fine-grained eleven gesture dataset recorded using the Soli sensor. While their approach shows a high average statistical accuracy of 87.17%, their proposed

model uses more than 600 MB of memory which is several orders of magnitude more than most low-power microcontrollers offer. Moreover, the Soli sensors are consuming more than 300 mW of power, which will drain any reasonably sized battery for a wearable device in a few minutes of use [82].

While it has been shown that TCNs can outperform LSTMs for action segmentation tasks both in terms of accuracy and inference speed [83, 98], the use of TCNs for gesture recognition remains a relatively unexplored field of research. However, one work by Luo et al. indicates that classical 2D-TCNs can perform equally well and even outperform approaches based on LSTM cells and HMMs for gesture recognition tasks [99].

2.1.3 Background

Range Frequency Doppler Map

Feature maps based on the Fourier transform of the time axis, like the Range Frequency Doppler Map (RFDM), similarly to micro-Doppler signatures, have been proven to be effective for machine learning applications in previous research on gesture recognition [94–97, 100]. It relies on the Doppler effect, which quantifies the shift of frequency in a signal that is reflected from a moving object. This shift of the frequency is correlated to the velocity of the object in the direction of the sensor. In order to detect changes in velocity, the I/Q signal is Fourier transformed into the frequency space, where changes in frequency can be observed. In order to detect the movement of objects in front of the sensor, multiple sweeps (i.e. time steps) are joined together and the time signal is Fourier transformed for each range point. As the sampled signal from each sweep $S(t, r)$ is time and range discrete the Discrete Fourier Transform (DFT) is used. The transformed feature map $S(f, r)$ can be calculated according to the following equation:

$$S(f, r) = \sum_{t=0}^{T} S(t, r) e^{-\frac{2\pi i f t}{T}}$$

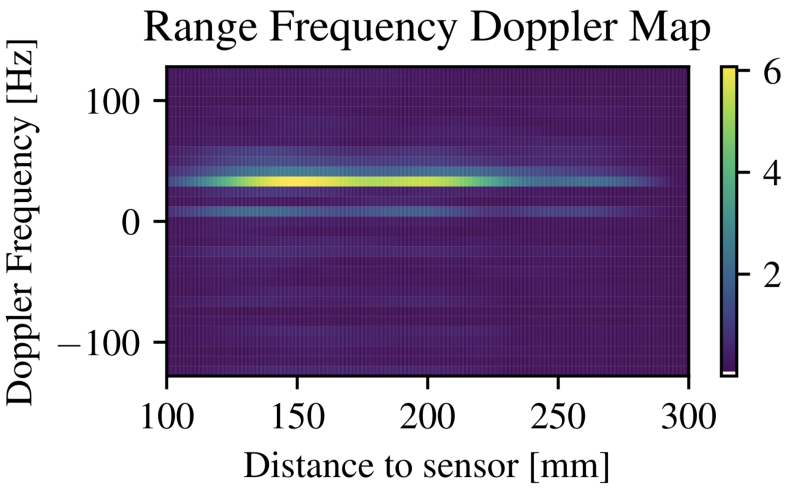

Figure 2.1: Range-Frequency Doppler Map for an example recording. The width and height dimensions correspond to the temporal frequency and the sampling range, respectively.

Where T is the total number of sample points per recorded distance point. We only consider the absolute values of this function. An example RFDM is shown in Figure 2.1.

Temporal Convolutional Networks

Temporal Convolutional Networks are a modelling approach for time series using dilated 1D-convolutional neural networks, proposed by Lea et al. [98], which has been used for a multitude of tasks, but very prominently in speech modelling [101, 102] and general human action recognition [103]. The basis of TCNs are causal, dilated 1D-convolutions. Causal refers to the fact that for the prediction of any time step no future inputs are considered. Thus, the support pixel of the kernel is always chosen to be the last pixel. This is needed in a real-time prediction scenario, as in that case only the current and

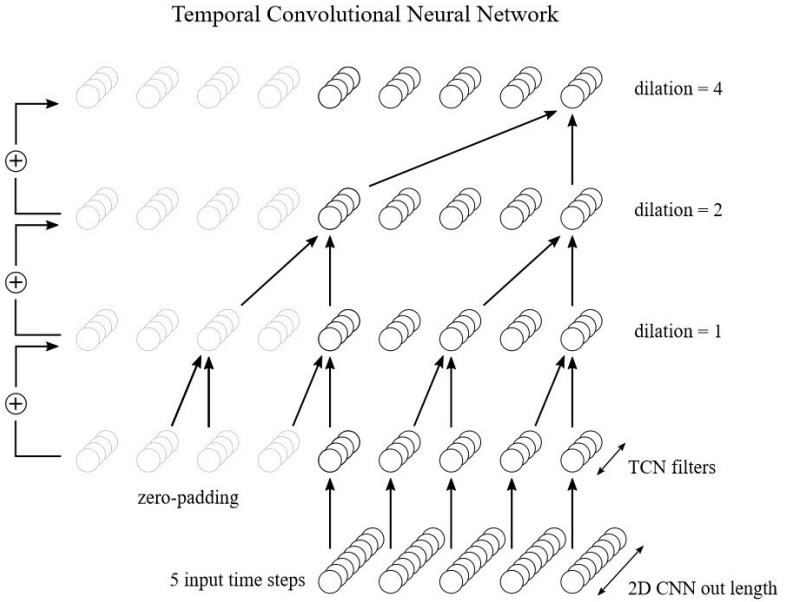

Figure 2.2: Layer structure of the TCN. Each input time step is one 1D vector that is generated by flattening the 2D CNN's output. The dilation factors used in the network are 1, 2 and 4. The kernel size for all convolutional filters in the TCN is 2.

past data values are available at prediction time. To weigh past data for sequence predicition, TCNs employ dilated convolutions over the temporal dimension. By increasing the dilation factor for consecutive layers the receptive field can be increased rapidly and very long effective memory of the network can be achieved. Figure 2.2 shows the data flow of the TCN. The input data for the TCN are the flattened, 1D outputs of a 2D-CNN.

Naturally, the TCN produces one output per time step. In the following, we will refer to metrics considering each individual time step as

per-frame and to metrics considering the time step and all previous time steps modelled in the TCN as *per-sequence*.

2.1.4 Low Power Short Range Radar and Dataset

This Subsection describes the properties of the Acconeer low power short-range RADAR sensor that was used and the parameters of the datasets that were acquired using the sensor.

Short Range RADAR for Gesture Recognition

The RADAR devices used are novel short-range pulsed Radio Detection and Ranging (RADAR) from Acconeer, pulsed with 60 GHz. These low power devices use only one transmitter and receiver which reduces the power consumption to tens of Milliwatts. The data returned by these sensors are sampled values of the I/Q signals. The RADAR sensor is configured to continuously emit pulses at a fixed frequency of f_{sweep}, called RADAR Repetition Frequency (RRF). The time interval between two pulses is called RADAR Repetition Interval (RRI).

Let $t = 0$ be the time at which the sensor sends out a pulse. Assuming that the transmitter and receiver are at the same position, i.e. being the same antenna, the response received at $t + 2\Delta t$ corresponds to the reflection echo of an object located at a distance of $d = \frac{c}{2\Delta t}$ from the emitter/receiver, where Δt is the time-of-flight of the pulse to the location of the object.
By regularly sampling the signal received after sending a pulse, a sweep vector containing reflections of objects at different distances can be computed. The distance resolution Δd of the Acconeer sensor amounts to 0.483 mm, which corresponds to a time-of-flight of 1.6 ns.

Dataset Specification and Acquistion

To train and evaluate the sensor for hand gesture recognition, two datasets were gathered: One 5-gesture dataset and two 11-gesture datasets.[6] The 11-gesture data set features the same gestures as Wang

[6]The 5G and 11G datasets and code for feature extraction are available for research purposes at https://tinyradar.ethz.ch

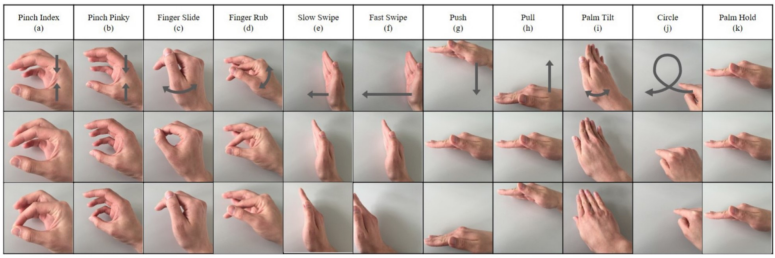

Figure 2.3: Overview of the gestures used in the dataset by Wang et al. [82]. The eleven gestures contain fine-grained gestures like "Finger Slide", as well as coarser gestures like "Push" or "Pull".

et al. [82] and the 5-gesture dataset uses a subset of the same 11 gestures, consisting of the "Finger Slide", "Slow Swipe", "Push", "Pull" and "Palm Tilt" gestures. Using the same gestures as Wang et al. [82] allows for an effective comparison. All eleven gestures are depicted in Figure 2.3.

The 11-gesture dataset uses two Acconeer sensors with a sweep rate of 160 Hz each, while the 5-gesture dataset uses a single sensor with a sweep rate of 256 Hz. Participants were shown Figure 2.3, the approximate height, 20 cm above the sensor board, at which to perform the gesture, but were given minimal instructions on how to perform the gestures. The gestures were performed in sitting position, without any additional inclination. The recording setup was not systematically varied between different persons and recordings. The 11-gesture dataset contains a total of 45 recording sessions of 26 different individuals, out of which 20 recordings are recorded from the same person to evaluate single-user accuracy, while the other 25 recordings are each recorded from different individuals. Subsets of the 11-gesture dataset are used to evaluate single user (SU) performance and multi-user (MU) performance. For the single-user dataset, the aforementioned 20 recordings from one single individual are used. For the multi-user dataset, one recording of the same individual is merged

Table 2.1: Overview of the parameters used to record the dataset

Parameters	**5-G**	**11-G (SU)**	**11-G (MU)**
Sweep frequency	256 Hz	160 Hz	160 Hz
Sensors	1	2	2
Gestures	5	11	11
Recording length	3 s	$\leq$ 3 s	$\leq$ 3 s
# of different people	1	1	26
Instances per Session	50	7	7
Sessions per recording	10	5	5
Recordings	1	20	26
Instances per gesture	500	710	910
Instances per person	2500	7700	35
Total Instances	2500	7700	10010
Sweep ranges	10 – 30 cm	7 – 30 cm	7 – 30 cm
Sensor modules used	XR111	XR112	XR112

with the remaining 25 recordings of different individuals, which results in a dataset of 26 recordings of 26 different individuals. Thus, the multi-user and single-user datasets overlap by one recording of one individual.

A complete overview of the dataset parameters can be found in Table 2.1.

2.1.5 Energy-Efficient and High Accuracy Gesture Recognition Algorithm

One of the major contributions of this Section is the proposal of a model to accurately classify hand gestures recorded with a short-range RADAR sensor. The proposed model enables the reduction of memory and computational resources, which pose the biggest challenge for the deployment of a model for small embedded devices such as microcontrollers.

The constraints for peak memory use and throughput were chosen to work with microcontrollers like the ARM Cortex-M7 series and RISC-V based devices with a power budget in the order of tens of milliwatts.

These microprocessors are very memory-constrained, usually offering below 512 KB of memory, and achieve optimal operating conditions when using 8-Bit quantization for the activations and 16- or 8-Bit quantization for the weights [104].

Preprocessing

Since the dataset consists of periodic samples of distance sweep vectors, we chose to use the well-known approach of stacking a number TW of sweep vectors into one feature map window of raw data, which is called a frame. For the proposed network, the number of sweep vectors was chosen to be 32. This corresponds to a total time resolution of 200 ms per frame for 11-G datasets and 125 ms for the 5-G dataset. These frames are then processed by normalizing them and computing their RFDM. While the 2D range-frequency spectrum contains a real and an imaginary component, only the absolute value of each bin is used, since the phase component of the spectral representation, while having the same number of values as the magnitude, did not add any significant improvement to the overall inference accuracy.

Neural Network Design

For the 11-G dataset, the input feature map size is $492 \times 32 \times 2$ values, as each sensor contributes one channel, the number of time steps considered are 32 and the number of range points per sweep is 492. Even when compressing each value to 8 bit, the total required buffer memory for each frame amounts to 246 KB. For successful time-sequence modelling, the information of multiple frames needs to be stored and processed. Using the raw frame for multiple time steps would lead to buffer space requirements in the order of megabytes, which is not available in commercial microcontrollers.

To solve this issue, the proposed model is based on a combination of a 2D CNN and a 1D TCN, which are designed to separate the spatial-temporal modelling problem into two parts; a short-term, spatial modelling problem, which captures little temporal information and can be solved on the level of individual frames, and a sequence modelling problem which can be solved on the level of extracted fea-

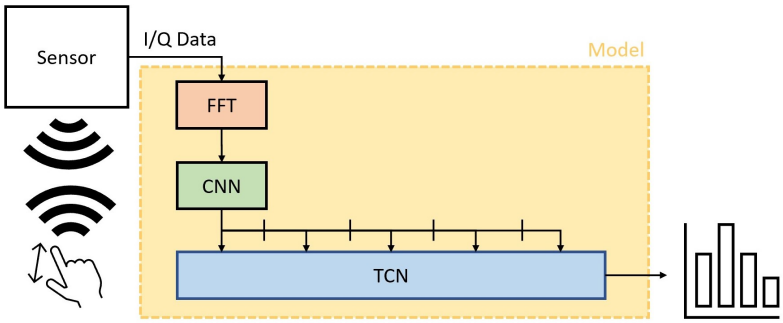

Figure 2.4: Overview of the processing algorithm. The raw I/Q sensor data is first processed by applying a Fourier transform, after which features are extracted from the frequency maps by processing them using a 2D CNN. The results of the feature extraction stage are flattened and five time steps are processed using a dilated TCN network.

tures from the first network. The overall data flow is depicted in Figure 2.4.

Spatial and Short-Term Temporal Modelling

Spatial and short-term temporal modelling can be seen as the task of extracting spatial and short-term temporal information from a single frame of RADAR data into a 1D feature vector containing spatial features that can be accurately classified with a sequence modelling algorithm. This approach compresses each frame by a factor of 82×, which allows the extracted features to be stored on the low-memory microcontrollers for multiple time steps, which is required for accurate time-sequence prediction. The proposed network for spatial feature extraction is depicted in Figure 2.5.

Since the width direction of the data frames corresponds to the spatial dimension, i.e. the distance from the sensor and the height direction corresponds to the temporal dimension of the frame, the frame width is considerably greater than the frame height. Since the distance

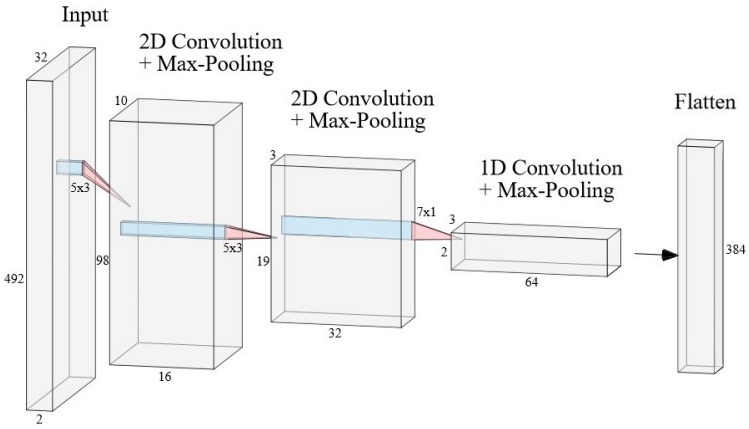

Figure 2.5: Layer structure of the 2D CNN. Each Convolutional layer is followed by a ReLU activation.

sampling is chosen to be very fine-grained, wide kernels are used, both for pooling and convolutions. The layer parameters are shown in Table 2.2. The total required buffer memory size for inference for algorithms using a static allocation of memory is given by the maximum of the sum of the buffer space required for the input and output feature map of any layer. For the proposed network, the total required buffer size is reached in the first layer and amounts to $(492 \cdot 32 \cdot 2 + 98 \cdot 10 \cdot 16) \cdot 8$ Bit = 368 KB.

Long-Term Temporal Modelling

The features computed by the 2D CNN are processed further with a TCN. The TCN uses an exponentially increasing dilation factor to combine features from different time steps into a single feature vector which can then be passed to a classifier consisting of fully-connected layers. For the proposed network, five time steps are considered by the TCN, i.e. five consecutive output feature vectors of the 2D CNN are used as the input of the TCN. This corresponds to a total effec-

Table 2.2: Layer architecture of the 2D CNN

Layer	Input	Output	Kernel	Padding
2D Conv	32×492×2	32×492×16	3×5	Same
Max Pooling	32×492×16	10×98×16	3×5	Valid
2D Conv	10×98×16	10×98×32	3×5	Same
Max Pooling	10×98×32	3×19×32	3×5	Valid
1D Conv	3×19×32	3×19×64	1×7	Same
Max Pooling	3×19×64	3×2×64	1×7	Valid
Flatten	3×2×64	384	-	-

tive time window of 1 s for the 11-G datasets and 0.625 s for the 5-G dataset. The overall TCN structure, taking into account the exponential dilation steps, is depicted in Figure 2.2.

Each TCN filter in the TCN consists of residual blocks, each consisting of one depthwise convolution layer followed by a ReLU [105] activation, the result of which is then added to the original input. This is slightly different from the original definition of residual blocks in Lea et al. [98], as normalization layers, dropout layers and one depthwise convolutional layer are removed to save memory space and execution time. A graphical comparison of the residual blocks as proposed by Lea et al. can be seen in Figures 2.1.5 and 2.1.5.

To reduce dimensionality, the output of the 2D CNN is filtered with a 1D Convolution which compresses the number of channels by a factor of 12×. The compressed features are then collected for a total of five time steps before being passed to the dilated network. For the final output classification, the output of the dilated network is passed to three fully-connected layers. The resulting network structure is described in Table 2.3.

Training Setup

Both the 2D CNN as well as the TCN were implemented using the Keras/Tensorflow framework. The RFDM features were extracted from the dataset and saved before training. Both network parts were trained together, using a batch size of 128 for a total of 100 epochs.

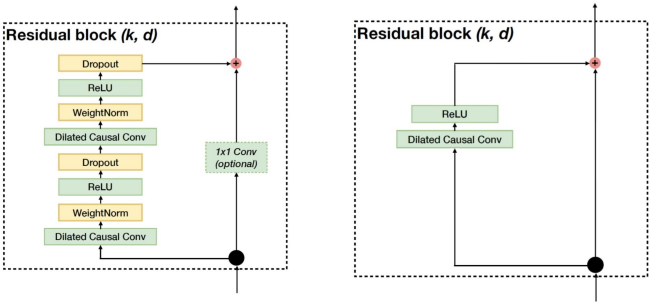

Figure 2.6: Comparison of the TCN residual blocks. The proposed network blocks (right) require a factor 2× less computations and memory than the original blocks (left), due to using only one convolutional layer instead of 2.

Table 2.3: Layer architecture of the TCN

Layer	**Input**	**Output**	**Kernel**	**Dilation**
Causal 1D Convolution	5×384	5×32	1	-
Causal 1D Convolution	5×32	5×32	2	1
Adding Layer	5×32	5×32	-	-
Causal 1D Convolution	5×32	5×32	2	2
Adding Layer	5×32	5×32	-	-
Causal 1D Convolution	5×32	5×32	2	4
Adding Layer	5×32	5×32	-	-
Fully connected	5×32	5×64	-	-
Fully connected	5×64	5×32	-	-
Fully connected	5×32	5×11	-	-

The optimizer chosen for training is Adam [106]. Both 5-fold cross-validation (CV5) and leave-one-user-out cross-validation (LOOCV) training runs were performed and are shown in the results Subsection (Subsection 2.1.6).

2.1.6 Results and Discussion

We evaluated the proposed model and its implementation on embedded hardware in terms of power consumption and inference performance on the system-scale. In particular, we present the test setup and the evaluation of the proposed model in terms of accuracy, memory and computational requirements in the first subsubsections, comparing different features and processing alternatives, while we present an evaluation of the implementation on a novel RISC-V-based parallel processor in a later subsubsection.

Experimental Setup

The GAP8 from Greenwaves Technologies[7] is an off-the-shelf RISC-V-based multicore embedded microcontroller developed for IoT applications. At its heart, the GAP8 features one RISC-V microcontroller and an octa-core RISC-V processor cluster with support for specialized DSP instructions, derived from the PULP open-source project [77]. The GAP8 memory architecture features two levels of on-chip memory hierarchy, containing 512 KB of L2 memory and 64 KB of L1 memory.

Figure 2.7 shows the hardware test setup, using evaluation boards for the GAP8 and A111 RADAR sensor, connected with an ARM Cortex-M4 evaluation board, which is used to broadcast the data to both a connected PC and the GAP8.

The trained model was deployed onto the GAP8 with the AutoTiler tool[8], which generates C Code optimized for parallel execution of the model on the hardware platform.

[7] https://greenwaves-technologies.com/ai_processor_gap8/

[8] https://greenwaves-technologies.com/manuals/BUILD/AUTOTILER/html/index.html

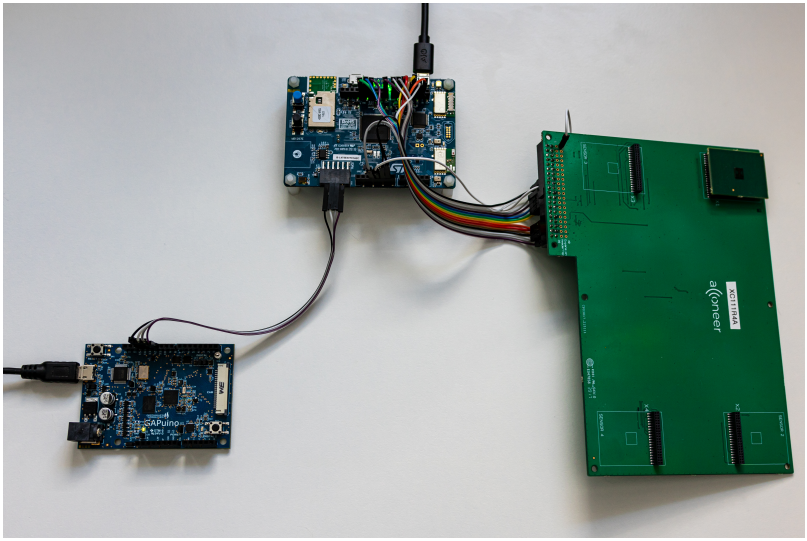

Figure 2.7: Picture of the hardware setup used to evaluate the system. The central board is a STM32L4 development board used to interface the RADAR sensor board (right) with the GAP8 development board (left).

Table 2.4: Per-frame and per-sequence inference accuracy of the full algorithm on the respective test/validation set

Metric	Per-Frame	Per-Sequence
5-G SU-CV5	93.83%	95.00%
11-G SU-CV5	89.52%	92.39%
11-G MU-CV5	81.52%	86.64%
11-G MU-LOOCV	73.66%	78.85%

Accuracy of the Algorithm

The inference accuracy of the algorithm can be discussed both in terms of per-frame accuracy, i.e. considering every frame for only one time step or in terms of per-sequence accuracy, i.e. the prediction for each frame taking into account the prediction for the individual frame at all time steps. To fairly compare results on the same dataset and frame definition, the per-frame metric is preferable, since it allows to accurately compare different approaches and the impact of sequence modelling versus single-frame processing. For comparing to other datasets and frame definitions, the per-sequence accuracy is the preferable metric, since it levels out the impact of using frames with higher time resolution and represents more accurately how the network behaves in a practical setting. The final results for the proposed network, both in terms of per-frame and per-sequence accuracy are shown in Table 2.4.

For the following paragraphs, the per-frame accuracy is used to discuss the impact of changes in architecture and pre-processing, while the per-sequence accuracy is used to compare to other research.

Evaluation of Pre-Processing Methods

To increase classification performance, different pre-extracted features were evaluated in combination with the features extracted by the convolutional neural network. The pre-extracted features are the signal energy, both for the Signal-over-Range (SOR) as well as the Signal-over-Time (SOT), the signal variation for the SOR and SOT and the centre of mass, which measures the intensity of the signal over

Table 2.5: Overview of the size of different input features

Feature	**Data Format**	**5-G**	**11-G**
Raw I/Q Signal	TW × RP × 2	26496	62976
Signal Variation 2D	(TW-1) × RP × 2	25668	61008
RFDM	TW × RP	13248	31488
Signal Energy SOR	RP	414	492
Signal Energy SOT	TW	32	32
Signal Variation SOR	RP	414	492
Signal Variation SOT	TW	32	32
Centre of mass	TW × 3	96	96

the range of the sensor. An important consideration for embedded systems is the size of the feature maps since memory is the most common bottleneck for neural network implementations on microcontrollers and similar devices. An overview of the number of values per feature with respect to the number of sampling windows TW and the number of range points RP can be found in Table 2.5.

Due to the splitting of the data into windows containing both spatial and temporal information, an evaluation of the preprocessing and pre-extracted feature performance using the 2D CNN and a fully-connected layer to estimate the feature quality can be given. Using this setup, the per-frame training accuracy results in Table 2.6 were achieved.

The RFDM features provide the best baseline in terms of pre-processed feature maps, both in terms of memory efficiency as well as classification performance. The raw data shows similar performance as the RFDM in the case of a single-frame model, which makes it important to consider as using the raw data needs no pre-processing, while all other features do. However, the required energy to calculate the RFDM features is around 34× less than what is used for one inference of the 2D-CNN, so the impact of pre-processing on energy efficiency is negligible. To further increase the accuracy, combinations of the RFDM with signal energy, variation and centre of mass were also studied. The per-frame performance of the RFDM features combined with other features can be seen in Table 2.7.

Table 2.6: Overview of the per-frame performance of different features for the 2D-CNN

Feature Combination	**5-G SU-CV5**	**11-G MU-CV5**
Raw I/Q Signal	90.35%	69.09%
Signal Variation 2D	89.93%	65.32%
RFDM	**91.08%**	**69.37%**
Signal Energy SOR & SOT	70.25%	51.90%
Signal Energy SOR	65.67%	49.95%
Signal Energy SOT	64.40%	40.72%
Signal Variation SOR	38.10%	17.92%
Signal Variation SOT	20.92%	10.57%
Centre of mass	47.56%	33.81%

Table 2.7: Overview of the 2D-CNN per-frame network performance with combined features

Feature Combination	**5-G SU-CV5**	**11-G MU-CV5**
RFDM baseline	91.08%	69.37%
RFDM & signal variation 2D	91.05%	**71.93%**
RFDM & signal energy SOR	90.99%	70.24%
RFDM & signal variation SOR	91.08%	69.16%
RFDM & centre of mass	**91.34%**	70.35%
RFDM & signal variation SOT	76.93%	59.33%
RFDM & signal energy SOT	91.20%	70.33%

Table 2.8: Overview of the averaged per-frame accuracy of the whole network with combined features

Feature Combination	**5-G SU-CV5**	**11-G MU-CV5**
Raw I/Q Signal	91.90%	76.91%
RFDM	**93.83%**	**81.52%**
RFDM & signal variation 2D	92.75%	78.84%
RFDM & signal energy SOR	93.22%	80.92%
RFDM & centre of mass	91.81%	78.45%
RFDM & signal energy SOT	93.38%	78.99%

As already shown in the evaluation of pre-processing methods, the added features do not increase accuracy by a significant margin, which substantiates the choice not to add them for the proposed network.

Hyperparameter Tuning of the TCN

The performance of the network with the added TCN was evaluated against the performance of the 2D CNN alone. As explained in Subsection 2.1.5, the number of TCN filters is independent of the rest of the network and can be tuned to fit the constraints of the application and target hardware. To find the optimal operating point for the number of filters, the correlation between the number of filters and the increase in accuracy was evaluated for the 11 gesture dataset and is shown in Figure 2.8.

As can be seen in the graph, the classification accuracy plateaus after 32 TCN filters. The averaged per-frame accuracy for different selections of features using 32 TCN filters and five time steps can be seen in Table 2.8.

As previously discussed in the evaluation of the pre-processing methods, adding manually extracted features does not positively impact the overall accuracy of the network.

Further, for all combinations of features, especially with respect to the 11 gesture multi-user dataset, the TCN improves the per-frame accuracy of the overall network by a significant margin.

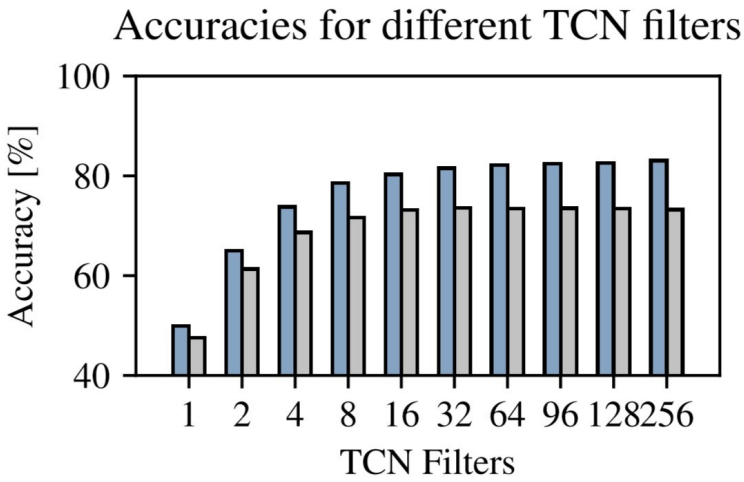

Figure 2.8: Classification performance vs. number of TCN filters on the 11-G dataset, using 5-fold cross-validation (blue) and leave-one-out cross-validation (grey). Even with exponential scaling of the number of filters, the accuracy stagnates after around 32 filters.

Table 2.9: Per-frame test accuracy of the whole network for different sequence modelling approaches using 32 filters

Time steps	**5**	**10**	**20**
LSTM, 32 filters	79.24%	79.69%	80.71%
LSTM, 128 filters	79.29%	80.23%	81.77%
Original TCN, 32 filters	80.50%	80.46%	81.49%
Original TCN, 128 filters	80.55%	80.26%	82.09%
Proposed TCN, 32 filters	80.13%	80.17%	81.45%
Proposed TCN, 128 filters	80.79%	81.32%	82.79%

Table 2.10: Number of parameters required for sequence modelling using LSTM vs. TCN broken down by number of filters

Filters	**32**	**64**	**96**	**128**
LSTM	25.4k	99.8k	223.5k	396.3k
Original TCN	12.4k	49.6k	111.2k	197.4k
Proposed TCN	6.2k	24.8k	55.6k	98.7k

Comparison to LSTM-based Networks

The proposed model's time-sequence modelling network using custom TCN layers was also evaluated against a modelling approach based on LSTMs as proposed by Schmidhuber et al. [79] and a network using standard TCN layers.

The performance for all three alternatives was evaluated using the same number of filters and time steps. The per-frame test accuracies for 32 and 128 filters are shown in Table 2.9.

The number of time steps beyond five does not significantly increase the inference performance of the network neither for the TCN version nor for the LSTM version. Besides accuracy, the focus for embedded deployment is always on network size. Table 2.10 shows the number of parameters for 32 and 128 filters. Note that the number of time steps does not impact the number of parameters.

The number of parameters for the TCN-based implementations is much lower than the number of parameters required for the LSTM-

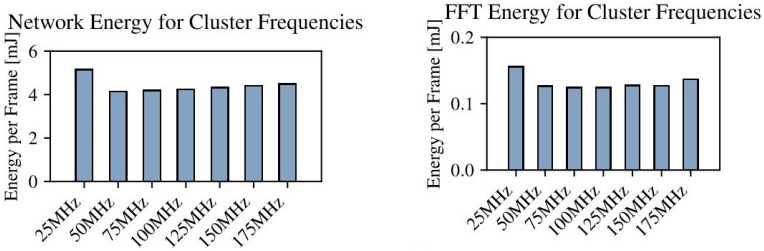

Figure 2.9: Overview of the microprocessor's energy efficiency while running the algorithm vs. its cluster frequency. To achieve real-time operation, at least 100 MHz are required.

based implementations. Taking into account the superior accuracy, smaller memory footprint achieved with the TCN-based implementations, the TCN models perform better by all evaluated metrics. Furthermore, using the proposed TCN variant, the number of parameters for the sequence modelling part can be reduced by a factor of 4× compared to LSTM-based variants.

Experimental Results

The proposed algorithm, as explained in Subsection 2.1.5, was implemented and evaluated on a GAPuino evaluation board and power measurements were taken for both the microcontroller as well as the RADAR sensor. The overall number of weights of the model is split between the 2D CNN, requiring 22'368 weights and the TCN, requiring 22'917 weights. Using 16 bit quantization and considering the implementation overheads, the network requires just under 92 KB on the GAP8. In terms of operations, the 2D CNN dominates the overall algorithm, taking up more than 99% of the overall computations, which total around 42 MOps per inference, taking a total of 5.8 MCycles per inference on the GAP8.

An overview of the energy consumption with respect to operating frequency is given in Figure 2.9.

Table 2.11: Energy breakdown of the algorithm on GAP8 at 100 MHz

Algorithm step	**Energy per Frame**	**Cycles**	**MACs**
FFT	0.12 mJ	$176 \cdot 10^3$	-
2D CNN	4.07 mJ	$5'100 \cdot 10^3$	$20'470 \cdot 10^3$
TCN	0.32 mJ	$458 \cdot 10^3$	$256 \cdot 10^3$
Dense	0.006 mJ	$86 \cdot 10^3$	$22 \cdot 10^3$
Full Network	4.52 mJ	$5'820 \cdot 10^3$	$20'750 \cdot 10^3$

Table 2.12: Power consumption of the RADAR sensor development board at different sweep frequencies

Sweep frequency	**Power consumption**	**Samples**
100 Hz	80 mW	300
160 Hz	95 mW	480
256 Hz	144 mW	768

For the system to work in real-time at 5 Hz prediction rate, including the sampling of the RADAR sensor and execution of the algorithm, the cluster frequency should be chosen to be at least 100 MHz. This leads to an average power consumption of 21 mW of the GAP8 microcontroller measured during 2 inference/sleep cycles, with peak power consumption of 98 mW while running the inference. An overall breakdown of operations, energy and cycles per inference at a clock frequency of 100 MHz using 8 cores is shown in Table 2.11.

To consider the overall system performance, the power consumption of the RADAR sensor has to be taken into account. Measuring the power consumption of the development board results in an upper bound, shown in Table 2.12.

Taking into account the power consumption for the RADAR sensors, we arrive at a system-level power consumption of around 200 mW when using two RADAR sensors at 160 Hz, and 115 mW when using one RADAR sensor at 160 Hz.

Table 2.13: Comparison with previous work

Metric	Soli	This work
Model size	689 MB	91 KB
Single sensor power consumption	300 mW	95 mW
Total sensor power consumption	300 mW	190 mW
Network inference power	–	21 mW
11-G SU Accuracy	94.5%	92.39%
11-G MU-CV5 Accuracy	-	86.64%
11-G MU-LOOCV Accuracy	88.27%	78.85%
Number of different users	10	26

Comparison to Previous Work

A direct comparison is most directly possible with previous work in Wang et al. [82] since we use the same set of gestures and evaluation metrics. In Table 2.13 we compare our results with those reported by Wang et al. All accuracies are reported per-sequence, as the definition of frames is different in [82].

The direct comparison shows that our proposed network performs comparably accurately, if slightly worse, in all but leave-one-subject-out cross-validation, to the network proposed by [82]. Nonetheless, our network size is smaller by a factor of 7'500× and our power consumption is lower by several orders of magnitudes, as [82] use a GPU for inference, which operates at tens to hundreds of Watts of power consumption.

2.1.7 Conclusion

We presented a high-accuracy and low-power hand-gesture recognition model combining a TCN and CNN model to achieve accuracy and low memory footprint. The model targets data processing with short-range RADAR. We further proposed a hand-gesture recognition system that uses low-power RADAR sensors from Acconeer combined with a GAP8 Parallel Ultra-Low-Power processor and can be battery operated. Two large datasets with 11 challenging hand-gestures performed by 26 different people containing a total of 20'210 gesture

instances were recorded, on which the proposed algorithm reaches an accuracy of up to 92.4%. The model size is only 92 kB and the implementation in GAP8 shows that live-prediction is feasible with a power consumption of the prediction network of only 21 mW. The results show the effectiveness and potential of RADAR-based hand-gesture recognition for embedded devices, as well as the network design, using the TCN approach. Further, we provide all necessary data and code to train the TinyRadarNN on tinyradar.ethz.ch.

2.2 WaveFormer: Long Sequence Transformers for Edge Devices

2.2.1 Introduction

Following the success of CNNs in the early 2010s, DNNs have become the standard approach for many machine learning applications, especially time-series processing [107]. While CNNs set records in statistical accuracy, they have recently been challenged by a newly emerging type of DNN architecture, the transformer [108]. The transformer was initially designed for use in NLP, but its main novelty, the attention mechanism, has proved helpful in various applications. Consequently, public benchmarks on open-source datasets are dominated by transformer network architectures in Automatic Speech Recognition (ASR) [109], speech separation [110], or environmental sound classification [111].

In parallel with this development, the increasing demand for machine learning at the edge has led to the cloud computing paradigm being challenged: most data centers are not equipped to process the amount of raw data generated by billions of devices fast enough, and battery-operated devices are not able to sustain the power needed to stream data continuously to the cloud for months or years without recharging [72]. Furthermore, energy and latency issues in cloud processing are compounded by privacy and safety concerns [112]. One solution to this dilemma is to perform machine learning inference directly on edge devices with limited memory and computational resources – an approach often referred to as *TinyML*.

The main requirement of TinyML, low inference latency enabling real-time computation, needs to be weighed against the constraints of MCUs, namely limited computational power, limited storage, and memory, typically in the order of hundreds of kilobytes, and the limited available energy in the context of always-on, battery-operated devices. The new generation of edge computing-focused MCUs are also able to process sub-word data efficiently by exploiting specialized SIMD ISA extensions, which help compensate for the low core

frequency of such devices and improve the energy efficiency of computation.

While research on TinyML systems has made significant progress in the past, the main class of networks studied remains CNNs. The attention layer is the main bottleneck preventing transformers' adoption for embedded time-series processing. The memory and computational requirements of the implementation of the conventional attention mechanism scale quadratically with the input length severely limiting the ability to process long sequences of data. A solution to this bottleneck is the use of alternative forms of attention [113–116]. While some of these works use heuristic approaches to reduce the complexity of the attention matrix [115], most rely on random feature maps to approximate the softmax kernel [113, 114, 116], without the costly explicit calculation of the attention matrix. The latter class of attention is typically referred to as *linear attention*.

This Section introduces the necessary building blocks to train and quantize full transformer models for deployment on MCUs with a power budget in the order of milliwatts. As a concrete high-impact application, this Section proposes the WaveFormer, an accurate and lightweight transformer-based neural network. Experimental evaluation shows that the WaveFormer model improves on the state-of-the-art in Keyword Spotting (KWS) datasets, namely the 12 and 35 class Google Speech Commands (GSC) datasets [117].

In detail, the main contributions of the Section are as follows:

- We introduce the WaveFormer, a neural network architecture that uses linear attention and processes raw audio data, achieving a new state-of-the-art accuracy of 98.8 % and 99.1 % on the GSC 12 and 35 class V2 dataset while requiring only 130 KParameters and 19 MOp per inference.
- We present a novel quantization methodology for neural networks that contain linear attention layers, which enables hardware-friendly integer quantization of linear attention networks. We apply this algorithm to the proposed network, demonstrating quantized inference without loss of statistical accuracy.

- We present hardware latency and power measurements of the quantized and deployed network on an Ambiq Apollo 4 Cortex-M4 microcontroller, achieving 741 ms of latency, enabling real-time operation, and energy consumption of 8.7 mJ per inference.
- We achieve 2.5 × lower memory and 4.7× lower computation requirements compared to the CNN-based state-of-the-art network on GSC 12 class dataset, as well as 8.5× lower memory and 4.2× lower computation requirements over the transformer-based state-of-the-art model on GSC 35 class dataset.

2.2.2 Related Work

As discussed in Section 2.2.1, time series classification tasks, such as keyword spotting, can largely benefit from the attention mechanism of transformer models. In the following, we briefly overview some of the most important works related to transformer models and keyword spotting in recent years.

Attention Mechanisms

The attention mechanism, as used in transformer literature, was introduced in Vaswani et al. [108], and refers to an operator that uses three tensor inputs, **q**(uery), **k**(ey), and **v**(alue), and produces a single tensor output. Each input tensor has a sequence length dimension **S** and an embedding dimension E, and is projected to an inner dimension **D**, using three trainable projections. The full dot-product attention mechanism is given by Equation 2.1.

$$\begin{gathered}\mathbf{Q}=\mathbf{q}\mathbf{W}_{\text{query}} \qquad \mathbf{K}=\mathbf{k}\mathbf{W}_{\text{key}} \qquad \mathbf{V}=\mathbf{v}\mathbf{W}_{\text{value}} \\ \text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) \doteq \mathbf{A}\times\mathbf{V} \doteq \underset{\text{over rows}}{\text{SoftMax}}\left(\frac{\mathbf{Q}\times\mathbf{K}^{\mathrm{T}}}{\sqrt{\mathbf{D}}}\right)\times\mathbf{V}\end{gathered} \tag{2.1}$$

The major bottleneck of this conventional attention mechanism is the computation of the $\mathbf{A}\times\mathbf{V}$ product, requiring $\mathcal{O}\left(\mathbf{S}^2\cdot\mathbf{D}\right)$ operations, where **S** refers to the sequence length, and **D** represents the inner dimension of the attention layer.

As an alternative to conventional attention, several works [113, 116] have proposed forms of *linear attention*, which, at its core, replaces the quadratic computation cost in sequence length of conventional attention with quadratic computation cost in the inner dimension, requiring $\mathcal{O}\left(\mathbf{S} \cdot \mathbf{D}^2\right)$ operations. Linear attention algorithms rely on applying the kernel trick with random feature maps Φ over the $\mathbf{Q}$ and $\mathbf{K}$ matrices to avoid calculating the softmax activation over the attention matrix $\mathbf{A}$. A possible formulation of the linear attention mechanism for each row i of the output matrix is given in Equation 2.2.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})\,[i] \approx \frac{\sum_{j=1}^{N} \Phi\left(\mathbf{Q}_i\right)^T \times \left(\Phi\left(\mathbf{K}_j\right) \times \mathbf{V}_j\right)}{\sum_{j=1}^{N} \Phi\left(\mathbf{Q}_i\right)^T \times \Phi\left(\mathbf{K}_j\right)} \tag{2.2}$$

While linear attention mechanisms do not offer any [113] or only asymptotical [116] equivalence with conventional attention, their computational complexity characteristics make them an exciting option for on-sensor applications like audio processing, where $\mathbf{S} >> \mathbf{D}$ if practical approaches can be found to achieve competitive accuracy.

TinyML for Keyword Spotting

Keyword Spotting is one of the most well-studied problems for TinyML systems, given its real-time, confidential near-sensor computation requirements. Early work by Zhang et al. [118] studied different recurrent, convolutional, and mixed neural network architectures, which establish a baseline for accuracy on the GSC and which drove further efforts towards efficient inference on MCUs [119]. Significant improvements were later achieved through the use of ResNet-based networks [120, 121], at the expense of increasing the models' memory and computational requirements.

More recently, transformer networks have been proposed [122, 123]. The transformers and convolutional networks for KWS share a common pre-processing approach, in which short-term spectrograms are stacked along the time dimension, resulting in a two-dimensional representation of the evolution of the short-term frequency spectrum. Because audio data tends to require sequence lengths in the order of

thousands but only one sampling dimension per microphone, conventional attention operations are too computationally intensive to be applied. This Section demonstrates how leveraging linear attention enables processing raw, uncompressed waveform data, significantly increases accuracy on keyword spotting tasks, and minimizes hardware requirements.

Transformer Quantization

An integral aspect of TinyML research is the quantization of neural networks to 8 or fewer bits. The basis of most quantization algorithms is PACT [16], which introduced trainable parameters for the clipping bounds of each tensor. Several works have extended the PACT algorithm, mainly focusing on the training convergence characteristics and initialization strategies of the clipping parameters [18, 124, 125].

While quantization algorithms for traditional CNNs are well studied [16, 18], 8-bit and mixed precision quantization algorithms, down to 2 bits [126], for standard Transformers have only recently been proposed [127–131]. To the best of the authors' knowledge, there is no previous work on quantized linear attention.

The proposed method in this Section achieves performance beyond the current state-of-the-art on embedded devices by training and quantizing a linear attention-based transformer model that operates on raw waveform data rather than pre-processed spectrogram windows. Our proposed network achieves an accuracy of 98.8 % and 99.1 % on the 12 and 35 class GSC V2 datasets and decreases model size and number of computations by 2.5× and 4.7× compared to state-of-the-art networks. We further present a novel, robust quantization methodology for linear attention networks, enabling efficient deployment to energy-efficient edge devices without loss of accuracy.

2.2.3 WaveFormer

One of the main contributions of this Section is the design of a KWS network, the WaveFormer. In contrast to other CNN- and transformer-based neural architectures for KWS [122, 123], the WaveFormer uses raw audio waveforms. To reduce the feature space,

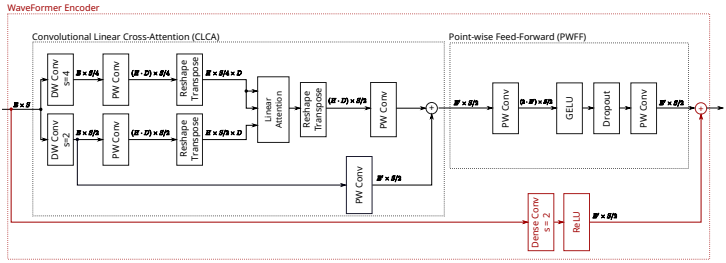

Figure 2.10: Overview of the encoder used in the WaveFormer. Each depthwise convolution uses a kernel size of 5, and each dense convolution uses a kernel size of 2. Shapes of intermediate tensors are annotated above the connecting arrows. H, S, E, and D represent the number of heads, sequence length, embedding dimension, and inner dimension, respectively.

the WaveFormer compresses the sequence length in each encoder block, reducing temporal resolution incrementally. This allows our model to learn a larger feature space, resulting in higher accuracy. To apply attention-based algorithms to long sequences found in audio recordings within the constraints on memory and computation imposed by embedded machine learning, the WaveFormer uses linear attention [113, 116].

Architecture

The main building block of the WaveFormer architecture is the Convolutional Linear Cross-Attention (CLCA) module. The CLCA implements Multi-Head Cross-attention [132] by using a convolutional projection for **Q** and a shared convolutional projection to compute the **K** and **V** matrices. Each convolutional projection consists of a depthwise convolution followed by a pointwise convolution. The depthwise convolution uses a configurable stride, which is used to reduce the sequence length. We found that adding a residual connection with a pointwise convolution between the downsampling depthwise convolution of the Q tensor and the output of the linear attention operator increases training stability. Following popular transformer literature

[108, 122, 123], we use a pointwise feed-forward module in sequence with the attention block. Finally, we found that a second residual connection with a strided dense convolution added to the output of the feed-forward layers increases the network's accuracy.

The configuration of hyperparameters used in our encoders is as follows: We use depthwise convolutions with a kernel size of 5 and a stride of 2 for the **Q** projection and a stride of 4 for the **K**,**V** projection, as our experiments showed that decreasing the stride of the **K**,**V** projection does not increase the accuracy of the network. In each attention block, we project the inputs to 2 heads, with an internal dimension that scales proportionally with the embedding dimension. The feature map activation function in the linear attention used in this Section, Φ, is a ReLU activation, similar to the generalized linear attention activation proposed in [113]. All feed-forward layers' [108] hidden dimension is equal to $2 \cdot E$, and we apply the commonly used GELU [133] activation. The dropout probability in the encoder is chosen as 0.1, as proposed in [108]. A block diagram of the WaveFormer encoder is shown in Figure 2.10.

Similar to KWT, LeTR, and other ViT-inspired architectures for conventional transformers [122, 123, 134], the full WaveFormer consists of repeated encoder modules. Each encoder module downsamples the sequence length and optionally expands the embedding dimension. Afterward, the feature tensor is averaged over the sequence dimension. The resulting vector is passed into a dense classifier, projecting the embedding dimension to the final classification label vector. The WaveFormer model configuration used throughout this Section consists of 9 sequential encoders with configuration as shown in Table 2.14.

Preprocessing

We resample the input of training, validation, and test inputs from 16.384 kHz to 8.192 kHz, shortening the sequence length. Since each sample in the GSC dataset is a one-second recording, the resulting input sequence length is 8192. We further apply random data augmentation in the form of polarity inversion (p = 0.8), Gaussian noise (p = 0.01), gain (p = 0.3), and reverb (p = 0.6) on the training

Table 2.14: Model architecture

Layer Type	Emb. Dim.	Seq. Len.	Head Dim.	MLP Dim.
Encoder	1	8192	4	8
Encoder	4	4096	4	16
Encoder	8	2048	8	16
Encoder	8	1024	8	32
Encoder	16	512	16	32
Encoder	16	256	16	64
Encoder	32	128	32	64
Encoder	32	64	32	128
Encoder	64	32	32	128
Sequence Avg. Pooling	64	16	-	-
Dense Layer	64	1	-	-

dataset. Finally, we quantize all inputs by scaling them linearly to the $(-128, 127)$ range, rounding to the nearest integer, and dividing by 128.

2.2.4 Quantization Algorithm

This section introduces a general linear quantization scheme for transformer models using linear attention layers to perform all tensor operations in integer arithmetic. For all operators in the network except linear attention and GELU activations, we use the TQT algorithm [18]. For GELU activations, we use the algorithm proposed in [128]. Building on this framework, we propose a novel quantization algorithm for linear attention focusing on the numerical stability of the division operator.

Quantized Linear Attention

As introduced in Section 2.2.2, the linear attention operator requires three fundamental operations, namely matrix multiplication, addition, and division. In this Section, we target only matrix multipli-

cations to be performed in 8-bit integer arithmetic since they form the bulk of computations in linear attention. In contrast, we do not perform the division operation with 8-bit integers but 32-bit integers, as most ISAs do not feature SIMD division extensions, and low precision division is numerically unstable. Our algorithmic contributions can be broken down into two core ideas: stabilization of the division operator and minimization of denominator rounding errors, which we introduce and motivate in this section.

Stabilization of Division

One core issue of performing divisions in integer arithmetic is avoiding division by zero. A common way to address this issue is adding a numerically small constant μ to the denominator, guaranteeing that the denominator is greater than zero. For our quantization algorithm, we have to take special care that the numerical value of μ is at least as large as the quantum of the denominator; otherwise, it will be rounded to zero, and division by zero might still occur. To this end, we introduce an integer parameter η, which acts as a scaling factor. As a quantization constraint, we enforce that $\eta \cdot \mu$ is larger than the denominator's quantum, resulting in Equation 2.3.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})\,[i] \approx \frac{\eta \cdot \sum_{j=1}^{N} \Phi\left(\mathbf{Q}_i\right)^T \times \left(\Phi\left(\mathbf{K}_j\right) \times \mathbf{V}_j\right)}{\eta \cdot \sum_{j=1}^{N} \Phi\left(\mathbf{Q}_i\right)^T \times \Phi\left(\mathbf{K}_j\right) + \eta \cdot \mu} \tag{2.3}$$

Denominator Rounding Error

A second issue of division is non-linear sensitivity to rounding errors in the denominator. While errors in the numerator of the division affect the output error linearly, rounding errors in the denominator for values close to zero affect the output error dramatically. To address the problem of asymmetric error sensitivity in the division operator during quantization, we apply percentile-based clipping to determine the initial clipping bound values of all quantized tensors. Initializing the clipping bounds based on a lower and upper percentile of the distribution, rather than the minimum and maximum, limits the dynamic range of quantized values and suppresses outliers. This increases the resolution of values closer to zero, which reduces the rounding error for

such values. Conversely, this clipping strategy increases the rounding errors of large values. Still, since they affect the output error of the division operator much less, percentile-based clipping increases the overall accuracy of quantized linear attention.

2.2.5 Deployment

To deploy networks to off-the-shelf microcontrollers, we developed DumpO, an extensible neural network deployment toolchain that targets microcontrollers. DumpO generates C code, which manages intermediate buffer memory and calls optimized kernel implementations of individual layers. Wherever possible, we used open-source CMSIS-NN [104] kernels and we used a custom kernel for GELU activations which implements the I-BERT [128] algorithm. Furthermore, we integrated a kernel for the quantized division described in Section 2.2.4.

As the deployment platform, we target the ARM Cortex-M4-based Apollo 4 microcontroller[9]. The Apollo 4 microcontroller supports core clock frequencies of 96 MHz and 192 MHz. It features three memory domains, of which two are SRAM-based and one is MRAM-based. The larger SRAM-based domain, the shared SRAM, features a total of 1 MB of memory, while the smaller domain, the TCM memory, features 384 kB. The MRAM features 2 MB of memory. To optimize execution speed on the microcontroller, we set up the linker to deploy the stack and heap on the TCM memory, which requires a single cycle for memory accesses. All code, model weights, and other data are stored in the MRAM and shared SRAM. The memories' access time is reduced with a hardware cache, which allows single-cycle access on hit. The cache miss penalty for the MRAM and SRAM regions are 11 and 7 cycles, respectively.

2.2.6 Experimental Results

Training Setup and Results

We performed all training experiments with a constant setup, described in this section. For both the 12 and 35 class problems, we

[9] https://ambiq.com/apollo4/

Table 2.15: Comparison of the WaveFormer model with state-of-the-art networks. Best results highlighted.

Model Name	Model Type	Model Size [Params]	Ops per Inference	Top-1 Acc. GSC-12-V2	Top-1 Acc. GSC-35-V2
CENet-GCN-6 [135]	Graph CNN	28 k	2.6 M	95.2 %	-
CENet-GCN-24 [135]	Graph CNN	56 k	9.1 M	96.5 %	-
CENet-GCN-40 [135]	Graph CNN	72 k	16.8 M	96.8 %	-
ConvMixer [136]	CNN	119 k	22.2 M	98.2 %	-
BC-ResNet-1 [121]	CNN	9 k	3.1 M	96.9 %	-
BC-ResNet-8 [121]	CNN	321 k	89.1 M	98.7 %	-
KWT-1 [122]	Transformer	607 k	53.8 M	98.1 %	97.0 %
KWT-3 [122]	Transformer	5360 k	526.3 M	98.6 %	97.7 %
LeTR-128s [123]	Transformer	404 k	40.4 M	-	97.9 %
LeTR-256 [123]	Transformer	1110 k	80.7 M	-	98.2 %
AST-Tiny [137]	Transformer	5800 k	782.2 M	98.1 %	97.7 %
Attention RNN [138]	Attention & LSTM	202 k	-	96.9 %	93.9 %
WaveFormer (This work)	Linear Transformer	130 k	19.0 M	**98.8 %**	**99.1 %**

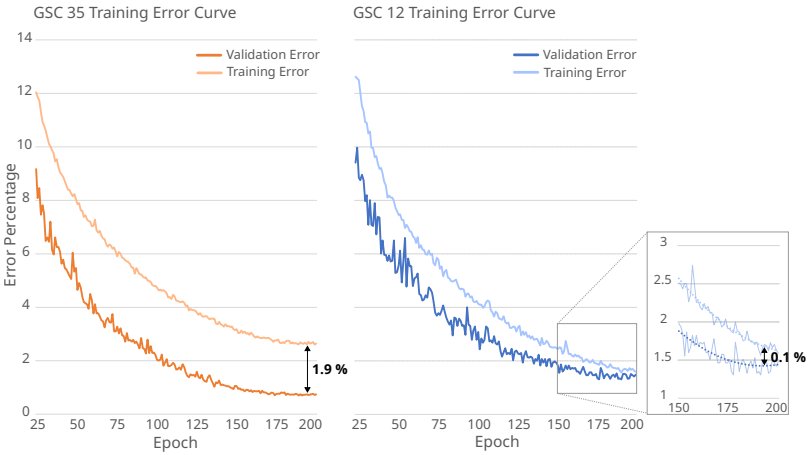

Figure 2.11: Plot of error rate curves for training and validation of the 35 class and 12 class model. Notably, the training-validation margin is much larger for the 35 class model, indicating better generalization.

train the model for 200 epochs using the Adam optimizer, with an initial learning rate of 2×10^{-3} with cosine-annealing learning rate scheduling. For the 12 class problem, we rebalance the class distribution for train, validation, and test sets using standard settings [121, 139]. Further, we average 20 training runs with different rebalancing split seeding. Since the datasets for the 35 class problem are fixed, we average four training runs with different seeds. For both problems, we use the train-validation-test split proposed in [117]. Notably, we evaluate the test accuracy on the standard test sets offered by the Google Speech Commands v2 dataset.

Using this setup, we measure an average accuracy of 99.1 % on the 35 class problem and 98.8 % on the 12 class problem. We also calculate the standard deviation for our experiments, achieving 0.114 for the 12 class problem, and 0.185 for the 35 class problem.

Unlike related work, we achieved higher accuracy on the 35 class problem. This is mainly due to two key reasons; first, we do not use

spectrogram compression on the input waveforms, which allows us to preserve a much richer feature space, which allows much better discrimination in the downstream network. Moreover, due to the rebalancing of the 12 class dataset, which discards most of the dataset, the 35 class dataset contains significantly more training data, enabling better network generalization, as shown by the training loss curves in Figure 2.11. To verify this hypothesis, we trained our proposed model on the 12 class problem without rebalancing the training set, thus the learning stage benefits from the same data as the 35 class problem, but with fewer categories. At the same time, the validation and test set maintain the sample per class balance as other works [121, 139], therefore ensuring a fair comparison with our method. In this experiment, we achieved an average accuracy of 99.7%, matching the results on the 35 class problem and outperforming existing work by 1 percentage point.

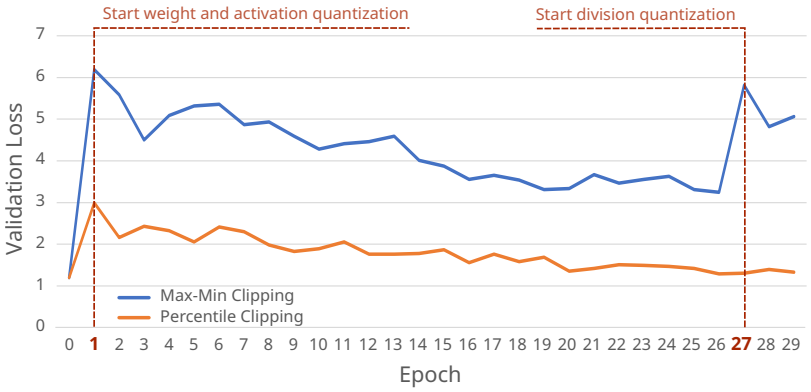

Figure 2.12: Validation loss curve of the 30 epoch quantization-aware training of a 12 class model. After an initial increase in loss, the percentile clipping quantization strategy recovers, while the max-min clipping strategy diverges once divisions are quantized.

Quantization

We used the Quantlib library[10] for all quantization experiments. Quantlib allows the import of a pre-trained network and the replacement of standard PyTorch layers with layers that implement fake-quantization algorithms and supports the export of fully integerized networks in a customized ONNX format. We extended the library to support percentile-based clipping initialization. Similarly, we extended the library with support for I-BERT quantization [128] of GELU layers. We used the library's implementation of the TQT [18] algorithm to train the clipping bounds for all layers.

We retrained the full-precision models with the Quantlib quantization-aware layer set for 30 epochs to achieve quantization-aware finetuning. We used an initial learning rate of 5×10^{-4} with cosine annealing scheduling. We start quantizing all convolutions, linear layers, and activations except for GELU after the first epoch and the quantization of GELU and Division layers after 27 epochs. We used the original train-validation-test split of the GSC V2 dataset and the same seed used for training for all quantization and retraining purposes. Furthermore, we disabled all dropout layers and kept the same data augmentation as during training.

Figure 2.12 shows the quantization training curves of a 12 class model, using the percentile-clipping-based strategy described in 2.2.4 and the default max-min-clipping strategy proposed in the original PACT paper [16]. While both clipping strategies expose an initial drop in accuracy after the first epoch, the clipping strategy converges to the original full-precision accuracy after a few epochs. In contrast, the initial decrease in accuracy for the max-min strategy is much more significant, and the network does not recover. The retrained, quantized 12 and 35 class networks using the percentile strategy achieve an accuracy of 98.7 %, and 99.2 % on the test dataset, closely matching their unquantized versions.

[10] https://github.com/pulp-platform/quantlib

Deployment Results

The generated implementation requires only 223 kB of Read-only memory for 152 kB of weights and 71 kB of program code and other overheads. Furthermore, the implementation requires 197 kB of Read/Write memory for intermediate buffers. On the Apollo 4 running at 192 MHz, one inference requires 142.3 MCycles for 19.8 MOp, leading to a compute intensity of 0.14 Op/Cycle and an inference latency of 741 ms for sequences of length 8192, while consuming an average of 11.7 mW. Since each input corresponds to a 1 s recording, this is sufficient for real-time operation with an energy cost of 8.7 mJ per inference.

Comparison with State-of-the-art

A comparison of our work with other networks on the GSC dataset is shown in Table 2.15. Notably, our proposed model achieves the highest absolute Top-1 accuracy reported in the literature for models of any type, both on the 12 class and the 35 class problem. The most closely comparable models for the 12 class problem in terms of accuracy, the BC-ResNet-8 [121] and KWT-3 [122] achieve 98.7 % and 98.6 % of accuracy on the test set, 0.1 and 0.2 percentage points less than our model. Comparing these models in terms of model size and number of operations, critical metrics for embedded machine learning, we find that our model uses 2.5× and 41× less parameters and 4.7× and 27.7× fewer operations per inference than the most accurate models reported in literature, BC-ResNet-8 and KWT-3 [121, 122]. Compared against the state-of-the-art for the 35 class problem, LeTR-256 [123], we achieve a 0.9 percentage point improvement in terms of accuracy, as well as a reduction in model size and number of operations of 8.5× and 4.2×. As mentioned in Section 2.2.6, we attribute these improvements to two key factors; Firstly, unlike all other models reported in literature, we do not use compressed spectrogram data but instead process raw waveforms, enabling deliberate and controlled compression of the model's feature space. Secondly, we use a novel form of linear attention, allowing us to efficiently process long sequences within the stringent memory and computational constraints of a microcontroller.

2.2.7 Conclusion

This Section presents a novel keyword spotting network architecture based on linear attention. We show that our model, which, in contrast to other state-of-the-art keyword spotting models, does not compress raw data before computation, outperforms the state-of-the-art in terms of accuracy by 0.1 and 0.8 percentage points while at the same time reducing the number of parameters by 2.5×, and the number of operations by 4.7×. We further propose a novel quantization scheme for linear attention layers and demonstrate quantization without loss of accuracy, enabling the use of the proposed network on resource-constrained embedded devices.

Chapter 3

CUTIE: Completely Unrolled Ternary Inference Engine

This Chapter presents a 3.1 POp/J fully digital hardware accelerator for ternary neural networks. CUTIE, the Completely Unrolled Ternary Inference Engine, focuses on minimizing non-computational energy and switching activity so that dynamic power spent on storing (locally or globally) intermediate results is minimized. This is achieved by 1) a data path architecture completely unrolled in the feature map and filter dimensions to reduce switching activity by favoring silencing over iterative computation and maximizing data reuse, 2) targeting ternary neural networks which, in contrast to binary NNs, allow for sparse weights which reduce switching activity, and 3) introducing an optimized training method for higher sparsity of the filter weights, resulting in a further reduction of the switching activity. Compared with state-of-the-art accelerators, CUTIE achieves greater or equal accuracy while decreasing the overall core inference energy cost by a factor of 4.8 ×–21 ×.

The following sections have been published in a slightly different form in the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems[1].

3.1 Introduction

Since the breakthrough success of AlexNet in the ILSVRC image recognition challenge in 2012 [1], CNNs have become the standard algorithms for many machine learning applications, especially in the fields of audio and image processing. Supported by advances in both hardware technology and neural network architectures, dedicated ASIC hardware accelerators for inference have become increasingly commonplace, both in datacenter-scale applications as well as in consumer devices [140]. With the increasing demand to bring machine learning to IoT devices and sensor nodes at the very edge, the *de facto* default paradigm of cloud computing is being challenged. Neither are most data centers able to process the sheer amount of data generated by billions of sensor nodes nor can typical edge devices afford to send their raw sensor data to data centers for further processing, given their very limited power budget [72]. One solution to this dilemma is to increase the processing capabilities of each sensor node to enable it to only send extracted, highly compressed information over power-intensive wireless communication interfaces or to act as an autonomous system.

However, the general-purpose microcontrollers typically employed in these IoT devices are ill-suited to the computationally intensive task of DNN inference, placing severe limitations on the achievable energy efficiency. While great strides in terms of energy efficiency have been made with specialized microcontrollers [77], some applications still require lower power consumption than what can be achieved with using 32-bit weights and activations in DNN inference. A popular approach to reducing the power consumption for neural network computations

[1] © 2021 IEEE. Reprinted, with permission, from M. Scherer, G. Rutishauser, L. Cavigelli, and L. Benini, "CUTIE: Beyond PetaOp/s/W Ternary DNN Inference Acceleration With Better-Than-Binary Energy Efficiency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1020–1033, Apr. 2022.

is the quantization of network parameters (weights) and intermediate results (activations). Quantized inference at a bit-width of 8 bits has been shown to offer equivalent statistical accuracy while allowing for significant savings in computation energy as well as reducing the requirements for working memory space, memory bandwidth, and storage by a factor of 4 compared to traditional 32-bit data formats [15, 141–143].

Pushing along the reduced bit-width direction, recently several methods to train neural networks with binary and ternary weights and activations have been proposed [144–149], allowing for an even more significant decrease in the amount of memory required to run inference. In the context of neural networks, binary values refer to the set {-1, +1} and ternary values refer to the set {-1, 0, 1} [144,150]. These methods have also been used to convert complex state-of-the-art models to their Binary Neural Network (BNN) or TNN form. While this extreme quantization incurs sizeable losses in accuracy compared to the full-precision baselines, such networks have been shown to work well enough for many applications and the accuracy gap has been reducing quite rapidly over time [151–153].

Although quantization of networks does not affect the total number of operations for inference, it reduces the complexity of the required multipliers and adders, which leads to much lower energy consumption per operation. For binary networks, a multiplier can be implemented by a single XNOR-gate [21]. Further, the number of bit accesses per loaded value is minimized, which not only reduces the memory footprint but also the required wiring and memory access energy.

While BNNs in particular are fairly well-suited to run on modern general-purpose computing platforms, to take full advantage of the potential energy savings enabled by aggressively quantized, specialized, digital, low-power hardware accelerators have been developed [21, 27, 154, 155]. Concurrently to the research in digital neural network accelerators, analog accelerators that compute in-memory, as well as mixed-signal, have been explored [26, 156, 157]. While mixed-signal and in-memory designs hold the promise of higher energy efficiency than purely digital designs under nominal conditions, their higher sensitivity to process and noise variations, coupled with the

necessity of interfacing with the digital world, are open challenges to achieve their full potential in energy efficiency [158].

Even though both analog and digital accelerators extract immense performance gains from the reduced complexity of each operation, there is still untapped potential to further increase efficiency. Most state-of-the-art binary accelerators use arrays of multipliers with large adder trees to perform the multiply-and-popcount operation [21, 26, 155, 159], which induces a large amount of switching activity in the adder tree, even when only a single input node is toggled. Adding to this, even state-of-the-art binary accelerators spend between 30% to 70% of their energy budget on data transfers from memories to compute units and vice-versa [26, 155]. This hurts efficiency considerably since time and energy spent on moving data from memories to compute units are not used to compute results. Taking these considerations into account, two major opportunities for optimization are to reduce switching activity in the compute units, especially the adder trees, and to reduce the amount of data transfer energy.

In this Chapter, we explore three key ideas to increase the core efficiency of digital low-bit-width neural network accelerator architectures: first, *unrolling of the data-path architecture with respect to the feature map and filter dimensions* leading to lower data transfer overheads and reduced switching activity compared to designs that implement iterative computations. Second, *focusing on TNNs instead of BNNs thereby capitalizing on sparsity to statistically decrease switching activity in unrolled compute units.* Third, *optimizing the quantization strategy of TNNs resulting in sparser networks that can be leveraged with an unrolled architecture.* We combine these ideas in CUTIE, the *Completely Unrolled Ternary Inference Engine.*

Our contributions to the growing field of energy-optimized aggressively quantized neural network accelerators are as follows:

1. We present the design and implementation of a novel accelerator architecture, which minimizes data movement energy spending by unrolling the compute architecture in the feature map and filter dimensions, demonstrating that non-computational energy spending can be reduced to less than 10% of the overall energy budget (Section 3.5.3).

2. We demonstrate that by unrolling each compute unit completely and adjusting the quantization strategy, we directly exploit sparsity, minimizing switching activity in multipliers and adders, reducing the inference energy cost of ternarized networks by 36% with respect to their binarized variants (Section 3.5.4).

3. We present analysis results, showing that the proposed architecture achieves up to 589 TOp/s/W in an IoT-suitable 22 nm technology and up to 3.1 POp/s/W in an advanced 7 nm technology, outperforming the state-of-the-art in digital, as well as analog in-memory BNN accelerators, by a factor of 4.8× in terms of energy per inference at iso-accuracy (Section 3.5.7).

This Chapter is organized as follows: in Section 3.2, previous work in the field of neural network hardware accelerators and aggressively quantized neural networks is discussed. In Section 3.3, we introduce the proposed accelerator architecture. Section 3.4 details the implementation of the architecture in the GlobalFoundries 22 nm FDX and TSMC 7 nm FF technologies. In Section 3.5, the implementation results are presented and discussed, by comparing with previously published accelerators. Finally, Section 3.6 concludes this Chapter, summarizing the results.

3.2 Related Work

In the past few years, considerable research effort has been devoted to developing task-specific hardware architectures that enable both faster neural network inference as well as a reduction in energy per inference. A wide range of approaches to increase the energy-efficiency of accelerators have been studied, from architectural and device-level optimizations to sophisticated co-optimization of the neural network and the hardware platform.

3.2.1 Aggressively Quantized Neural Networks

On the algorithmic side, one of the main recent research directions has been quantization, i.e. representing model weights and intermediate activations in lower arithmetic precision. It has been known

for some time that quantization of network weights to 5 bits and less is possible without a loss in accuracy in comparison to a 32-bit floating-point baseline model [15, 141, 142]. Further quantization of network weights to binary or ternary precision usually results in a small drop in accuracy, but precision is still adequate for many applications [147,148,160,161]. Extending the approach of extreme quantization to intermediate activations, fully binarized and fully ternarized networks have been proposed [144,150]. These types of networks perform very well on easier tasks such as 10-class classification on the well-established MNIST dataset [162], and efforts have been taken to improve their performance with novel training approaches [163, 164]. Nevertheless, on more challenging tasks such as classification on the ILSVRC'12 dataset, they are still significantly less accurate than their full-precision counterparts [145, 146, 149, 152, 165–167]. Figure 3.1 depicts the accuracy gap between previously published, strongly quantized neural networks, their full-precision equivalents with identical architectures and the state-of-the-art full-precision networks on image classification tasks of increasing difficulty. On higher difficulty tasks, the gap between quantized networks and their full-precision equivalents grows larger. Furthermore, the gap between the full-precision architectures from which the quantized networks are derived and the overall state-of-the-art results reported in literature grows with task difficulty, indicating a prevalent focus in research activity on easier tasks and simple networks.

Taking all of this into account, BNNs and TNNs provide a unique and interesting operating point for embedded devices, since they are by definition aggressively compressed, allowing for deep model architectures to be deployed to highly memory-constrained low-power embedded devices.

The core idea of binarization and ternarization of neural networks has been applied in numerous efforts, some of which also study the impact of the quantization strategy on the sparsity of ternary weight networks [148,176–178]. While these previous efforts focus on the impact of the choice of quantization threshold and regularization, we evaluate the impact of quantization order, rather than threshold or regularization. Further, we study the effect of sparsity on the energy-efficiency of the proposed accelerator architecture.

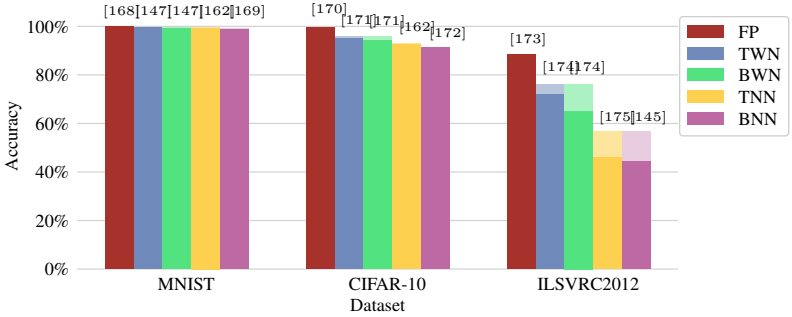

Figure 3.1: Comparison of state-of-the-art accuracy of highly quantized neural networks of different precisions. **FP**: state-of-the-art in unquantized/full-precision neural networks, **BWN/TWN**: binary/ternary weight networks, **BNN/TNN**: fully binarized/ternarized neural networks. For the quantized network categories, the accuracy of the corresponding unquantized baseline networks is shown greyed out. As task difficulty is increased, a) the performance gap between the quantized networks and the full-precision baselines increases, and b) the gap between the unquantized baselines from which the quantized architectures are derived and the full-precision state-of-the-art widens.

3.2.2 DNN Hardware Accelerators

While the first hardware accelerators used for neural networks were general-purpose GPUs, there has been a steady trend pointing towards specialized hardware acceleration in machine learning in the past few years [179–182]. Substantial research efforts have focused on exploring efficient architectures for networks using activations and weights with byte-precision or greater, [7,27,183] different digital ASIC implementations for binary weight networks and BNNs have been proposed [21, 143, 154, 155, 184, 185]. Some works have tackled analog ASIC implementations of TNN accelerators, [156,186], but very few digital implementations for TNN accelerators have been published [187,188].

At the heart of every digital neural network accelerator lie the processing elements, which typically compute Multiply-Accumulate (MAC) operations. An important distinction between different architectures, besides the supported precision of their processing elements, lies in the way they schedule computations [179]. Most state-of-the-art architectures can be categorized into systolic arrays [4,27,156,184], which are flexible in how their processing elements are used, or output-stationary designs, which assign each output channel to one processing element [155,159,179]. Both approaches trade-off lower area for lower throughput and increased data transfer energy by using iterative decomposition since partial results need to be stored and either weights or feature map data need to be reloaded. The alternative to iterative decomposition pursued in our approach, i.e. fully parallelizing the kernel-activation dot-products, is not only generally possible for convolutional neural networks, but also promises to be more efficient by increasing data-reuse and parallelism.

The state-of-the-art performance in terms of energy per operation for digital BNN and TNN accelerators is reported in Moons et al. [155] and Andri et al. [21], achieving peak efficiencies of around 230 TOp/s/W for 1-bit operations, as well as Knag et al. [159], reporting up to 617 TOp/s/W. The state-of-the-art for ternary neural networks is found in Jain et al. [156], achieving around 130 TOp/s/W for ternary operations.

In this work, we move beyond the state-of-the-art in highly quantized acceleration engines by implementing a completely unrolled data path. We show that by unrolling the data path, sparsity in TNNs is naturally exploited to reduce the required energy per operation without any additional overhead, unlike previous works [183,189–191]. To capitalize on this effect, we introduce modifications to existing quantization strategies for TNNs, which are able to extract 53% more sparsity at iso-accuracy than by sparsity-unaware methods. Lastly, our work shows that ternary accelerators can significantly outperform binary accelerators both in terms of energy efficiency as well as statistical accuracy.

3.3 System Architecture

This Section introduces the proposed system architecture. First, we present the data path and principle of operation and explain the levels of data re-use that the architecture enables, then we discuss considerations for lowering the overall power consumption. Finally, we present the supported functionality.

3.3.1 High-level Data Path

Figure 3.2 shows a high-level block diagram of the accelerator architecture. It is optimized for the energy-efficient layer-wise execution of neural networks. This is achieved first and foremost by a flat design hierarchy; each output feature map is computed channel-wise by dedicated compute units, called Output Channel Compute Unit (OCU). Each OCU is coupled with a private memory block for weight buffering, which minimizes addressing and multiplexing overheads for weight memory accesses, reducing the amount of energy spent on data transfers. The feature map storage buffers are shared between all OCUs to maximize the re-use of loaded activation data, which again aims to decrease the data transfer energy.

To exploit the high rate of data re-use possible with CNNs, the design uses a tile buffer, which produces tiles, i.e. square windows, of the input feature map in a sliding window manner. These windows are then broadcast to the pipelined OCUs.

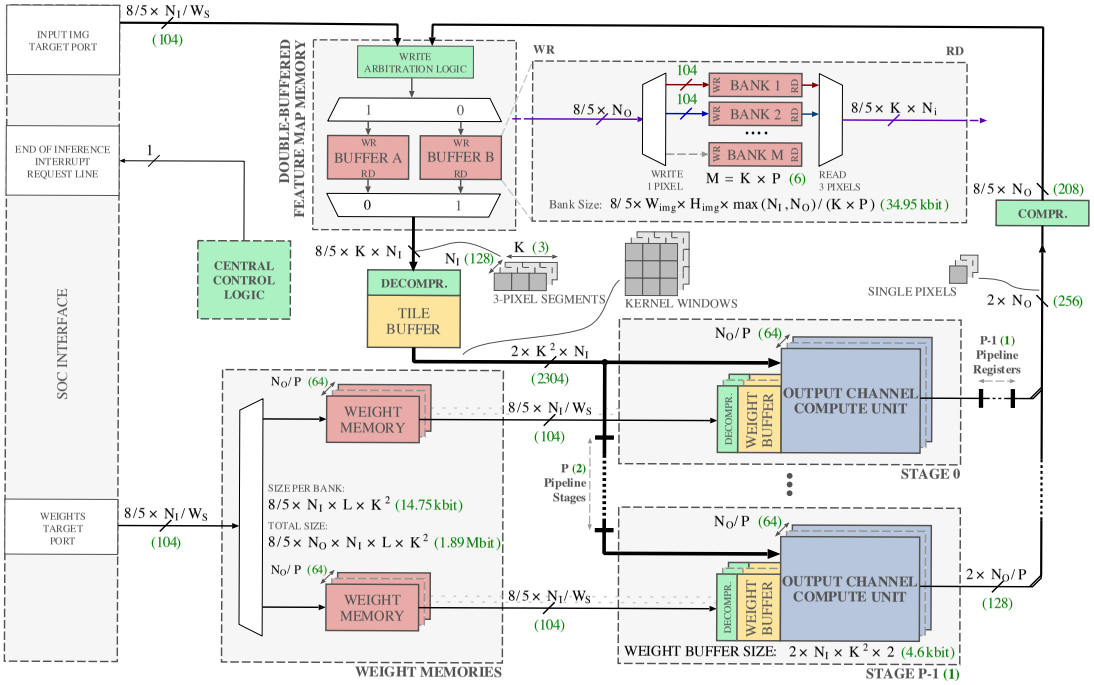

Figure 3.2: Data-path schematic view of the accelerator core and its embedding into an SoC-level system. The diagram shows the unrolled compute architecture and encoding/decoding blocks, as well as the weight and feature map memories and tile buffer module.

An important aspect of aggressively quantized and mixed-precision accelerator design is choosing a proper compression scheme for its values. Since ternary values encode $\log_2(3) \approx 1.585$ bits per symbol, the most straight-forward compression approach would require 2 bits of memory per value, leaving one of the four possible codewords unused. To reduce this overhead, values are stored 5 at a time, using 8 bits leading to 1.6 bits per symbol. The compression scheme used for this representation is taken from a recent work by Muller et al. [192]. To transition between the compressed representation and the standard 2's complement representation, compression and decompression banks are used with feature map and weight memories.

Figure 3.2 shows the pipeline arrangement of the OCUs. A key feature of the architecture is that an output channel computation is entirely performed on a single OCU. All OCUs need to receive input activation layers: the broadcast of input activations to OCUs is pipelined and the OCUs are grouped in stages. This pipeline fulfils multiple purposes: from a functional perspective, it allows to silence the input to clusters of compute units, which reduces switching activity during the execution of layers with fewer output channels than the maximum. Concerning the physical implementation of the design, pipelining helps to reduce fanout, which further reduces the overall power consumption of the design. It also reduces the propagation delay introduced by physical delays due to long wires.

3.3.2 Parametrization

The CUTIE architecture is parametrizable at compile time to support a large variety of design points. An overview of the design parameters is shown in Table 3.1. Besides the parameters in Table 3.1, the design's feature map memories and weight memories can be implemented using either Standard Cell Memorys (SCMs) or SRAMs. CUTIE is designed to support arbitrary odd square kernel sizes K, pipeline depths P, input channel numbers N_I and output channel numbers N_O which directly dictate the dimensioning of the compute core, but also of the feature map memories and the tile buffer. The OCU, as shown in Figure 3.4, consists of a compute core and a latch-based weight buffer that is designed to hold two kernels for the computation of one

Table 3.1: Design parameters of CUTIE

Parameter	Description
N_I	Maximum number of channels of input feature map
N_O	Maximum number of channels of output feature map
K	Maximum kernel width and height
I_W	Maximum width of input feature map
I_H	Maximum height of input feature map
L	Maximum number of layers in the queue
P	Number of pipeline stages
W_S	Number of memory words per pixel

output channel, which amounts to $4 \times K^2 \times N_I$ bits. The feature map memories are designed to support the concurrent loading of K full pixels as well as the granular saving of $\frac{N_O}{P}$ ternary values. For these reasons, the word width of the feature map memories is chosen to be $\frac{N_O}{P}$ ternary values. To further allow for concurrent write and read accesses of up to K pixels, two feature map memories, each with $P \times K$ feature map memory banks, are implemented.

3.3.3 Principle of Operation

The accelerator core processes neural networks layer-wise. To enable layer-wise execution, networks have to be compiled and mapped to the core instruction set. The compilation process achieves two main goals: first, the networks' pooling layers are merged with the convolutional layers to produce fused convolutional layers. Second, the networks' convolutional layers' biases, batch normalization layers, and activation functions are combined to produce two thresholds that are used to ternarize intermediate results, similar to constant expression folding for BNNs [185]. After compilation, each layer consists of a convolutional layer with ternary weights, followed by optional pooling functions and finally, an activation function using two thresholds that ternarizes the result. To map the network to the accelerator, each layer's weights are stored consecutively in the weight memories, the thresholds are stored consecutively in the OCUs' Threshold FIFO and

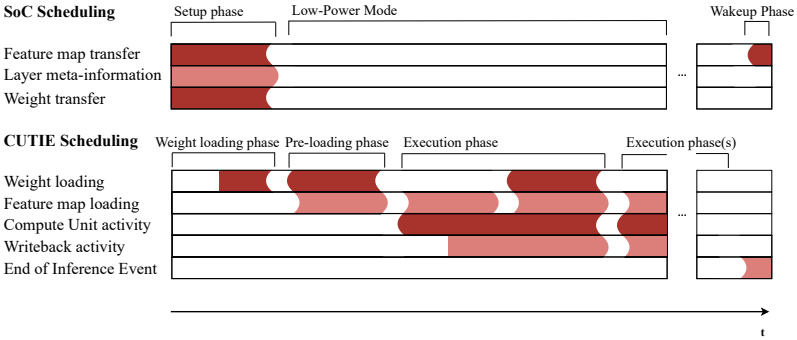

Figure 3.3: Scheduling diagram of the accelerator core and SoC interface. The first two phases are needed to set up the first layer after reset, every other loading phase overlaps with an execution phase, which reduces the latency for scheduling a new layer to a single cycle. The host system can be put in a low-power mode while the accelerator core computes the network since all layer information is saved inside the core's memories.

the meta-information like input width, stride, kernel size, padding, and so on are stored in the layer FIFO. All FIFOs, controllers and scheduling modules combined make up 2% of the total area.

The accelerator is designed to pre-buffer the weights for a full network during its setup phase and re-use the stored weights for multiple executions on different feature maps. Once at least one layer's meta-information is stored and the start signal is asserted, the accelerator's controllers schedule the execution of each layer in two phases; first, the weights for one layer are loaded into their respective buffers in the OCUs, then the layer is executed, i.e. every sliding window's result is computed and written back to the feature map memory. The loading of weights into the OCUs for the next layer and the computation of the current layer can overlap, leading to a single, fully concurrent execution phase after buffering the first set of weights, as shown in Figure 3.3. Once all layers have been executed, the end of inference signal is asserted, signalling to the host controller that the results are valid and the accelerator is ready for the next feature map input.

The module responsible for managing the loading and release of sliding windows is the tile buffer. The tile buffer consists of a memory array that stores K lines of pixel values implemented with standard cell latches. Feature maps are stored in a (H×W×C)-aligned fashion in the feature map memory. To avoid load stalls and efficiently feed data to the compute core, up to K adjacent pixels at a time are read from the feature map memory. The load address is computed to always target the leftmost pixel of a window.

The scheduling algorithm for the release of the windows keeps track of the central pixel of the next-to-be scheduled window. This can be used to enable padding: for layers where padding is active, the scheduler starts the central pixel at the top left corner and zero-pads the undefined edges of the activation window. In case of no padding, the scheduler starts the central pixel to the lower-right of the padded starting position. For all but the first layer in a network, the weight loading and computation phases overlap such that the weights for the next layer are pre-loaded to eliminate additional loading latency.

The OCUs form the compute core of the accelerator. Figure 3.4 shows the block diagram of a single OCU. Each OCU contains two weight

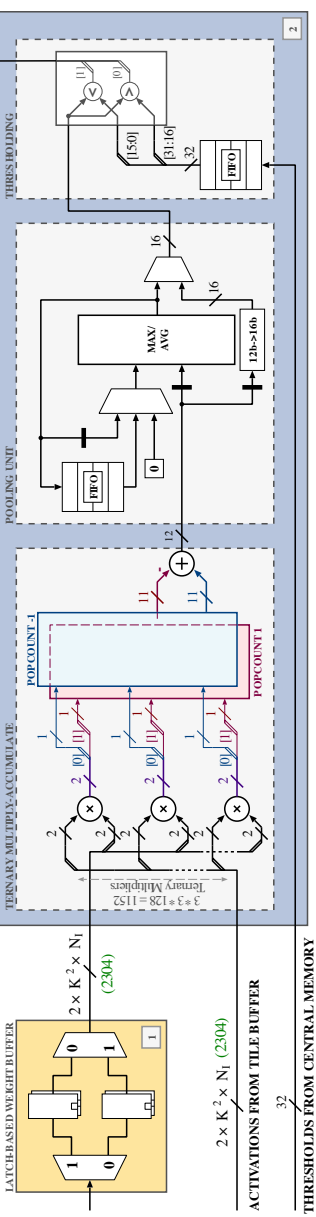

Figure 3.4: Block diagram of the compute units for the design point $K = 3$, $N_I = N_O = 128$, showing the dual inner weight buffers (1), used for double buffering to avoid load stalling, the OCU (2), including the completely unrolled multiply/add tree, computing 1'152 multiply-accumulate operations in a single cycle, the pooling block, which enables max and average pooling and the thresholding module used to ternarize intermediate results. Notably, the multiplier and popcounts are fully combinational and not pipelined, which adds to the energy efficiency of the compute core.

buffers, each of which is sized to hold all the kernel weights of one layer. Having two buffers allows executing the current layer while also loading the next layer's weights. The actual computations are done in the ternary multipliers, each of which computes one product of a single weight and activation. While the input trits are encoded in the standard two's complement format, the result of this computation is encoded differently, i.e. the encoding is given by f:

$$f(x) = \begin{cases} 2'b10 & x = 1 \\ 2'b01 & x = -1 \\ 2'b00 & x = 0 \end{cases}$$

This encoding allows calculating the sum of all multiplications by counting the number of ones in the MSB and subtracting the number of ones in the LSB of all results, which is done in the popcount modules. The resulting value is stored as an intermediate result, either for further processing with the pooling module or as input for the threshold decider. The threshold decider compares the intermediate values against two programmable thresholds and returns a ternary value, depending on the result of the comparison. Notably, the OCU is almost exclusively combinational, requiring only one cycle of latency for non-pooling layers. Registers are only used to silence the pooling unit and in the pooling unit itself to keep a running record of the current pooling window. Since every compute unit computes one output channel pixel at a time, there are no partial sums that have to be written back.[2] However, to support pooling, each compute unit is equipped with a FIFO, a register, and an Add/Max ALU. In the case of max pooling, every newly computed value is compared to a previously computed maximum value for the window. In the case of average pooling, values are simply summed and the thresholds that are computed offline are scaled up accordingly. Figure 3.5 shows an example of the load & store schedule for pooling operations.

Low-power optimizations have been made on all levels of the design, spanning from the algorithmic design of the neural networks over the system architecture down to the choice of memory cells.

[2] Which is a major difference from systolic arrays as well as output stationary designs!

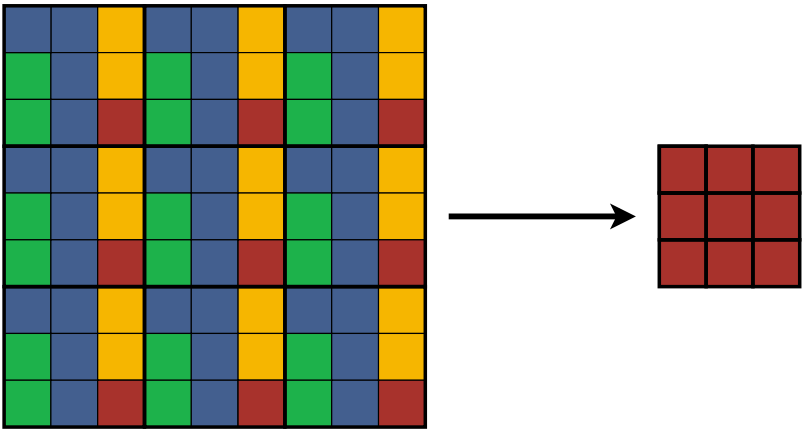

Figure 3.5: Example of pooling buffer scheduling for 9×9 feature maps applying 3×3 pooling. The feature map is traversed left- to-right, top-to-bottom. Blue pixels are stored in the pooling unit's register, yellow pixels are stored in the pooling unit's FIFO for later use and green pixels are loaded from the pooling unit's FIFO and compared to the current value. Best viewed in color.

```
1 for w in range(featuremap_width):
2  for h in range(featuremap_height):
3    for co in range(output_channels):
4      for ci in range(input_channels):
5        for kw in range(kernel_width):
6          for kh in range(kernel_height):
7            out_fm[w][h][co] += in_fm[w+kw][h+kh][ci]
8            * kernel[kw][kh][ci][co]
```

Listing 3.1: Loop unrolling of convolutional layers implemented in the CUTIE architecture. The highlighted lines 3-8 are computed in parallel in a single shot, in combinational logic. Each OCU computes one output pixel channel value, i.e. each OCU computes one instance of the third loop.

Unlike most state-of-the-art architectures which use either systolic arrays or output-stationary scheduling approaches with iterative decomposition [27, 155, 156, 159, 179, 184, 185], the CUTIE architecture unrolls the compute architecture fully with respect to weight buffering and output pixel computation, such that no storing of partial results is necessary; each output channel value is computed in a single cycle, as shown in Listing 3.1. The proposed design loads each data item exactly once and reduces overheads in multiplexing by clock gating unused modules. This applies to both the system level, with pipeline stages of the compute core that can be silenced, as well as to the module level, where the pooling module can be clock gated. To reduce both leakage and access energy, the feature map and weight memories can be implemented with standard cell latches, which are clock-gated down to the level of individual words. Generally, all flip-flops and latches in the design are clock-gated to reduce power consumption due to clock activity.

3.3.4 Input Encoding

To run real-world networks on the accelerator, the integer-valued input data has to be encoded with ternary values. We designed a novel ternary thermometer encoding based on the binary thermometer encoding [193]. The binary thermometer encoding is an encoding func-

tion f, that maps an integer between 0 and M to a binary vector with M entries.

$$f\colon \mathcal{N}_M \to \mathcal{B}^M$$
$$x \mapsto f(x)$$

$$f(x)_i = \begin{cases} 1 & i < x \\ -1 & i \geq x \end{cases}$$

The ternary thermometer encoding is an encoding function g that maps an integer between 0 and 2M to a ternary vector of size M.

$$g\colon \mathcal{N}_{2M} \to \mathcal{B}^M$$
$$x \mapsto g(x)$$

$$g(x)_i = \mathrm{sgn}(x - M) \cdot \frac{f(|x - M|)_i + 1}{2}$$

The ternary thermometer encoding makes use of the additional value in the ternary number set with respect to the set of binary numbers and can encode inputs that are twice the size for a binary vector of a given size. The introduction of 0s in the encoding scheme further helps to reduce toggling activity in the compute units, lowering the average energy cost per operation. As an example, for $M = 128$, and $x = 110$ the binary thermometer encoding produces $[1]^{110}\,[-1]^{18}$, whereas the ternary thermometer encoding produces $[-1]^{18}\,[0]^{110}$.

3.3.5 Exemplary Instantiations of CUTIE

The architecture of CUTIE is highly parametric. In the following, we present two practical embodiments of the general architecture, which we will then push to full implementation. The instantiations of the accelerator presented in this Section can process convolutions with a kernel of size 3×3 or smaller, using a stride between (1,1) and (3,3) with independent striding for the width and height dimension. It further supports average pooling and maximum pooling. Both no padding and full zero-padding, i.e. padding value of size 1 on every edge of feature maps, are supported. Depending on the requirements

of the application, the feature map memory size and weight memory size should be configured to store the largest expected feature map and network. For the sake of evaluating the architecture, we chose to implement one version that supports feature maps up to a size of 32×32 pixels for both the current input feature map and the output feature map using SCMs and another version supporting sizes up to 160×120 feature map pixels using SRAMs. The supported feature map memory size does not restrict the functionality, since feature maps that do not fit within the memory can be processed in tiles. Assuming the feature maps need to be transfered from and to an external DRAM memory which requires 20 pJ/Bit, several orders of magnitude more energy than accessing internal memory, the critical goal is to minimize the amount of data transfered from and to external memory. To achieve that, we propose to adopt the depth-first computing schedule described in [194].

To estimate the energy cost of processing the feature map in tiles and to compare the layer-first and depth-first strategies on CUTIE, we compute the number of processed tiles per layer, the number of tiles that need to be transfered over the chip's I/O and the number of weight kernels that need to be switched for both the depth-first as well as the layer-first strategies. We assume a network consisting of eight convolutional layers using 3×3 kernels and 128 input and output channels. Using these results and simulated energy costs for computations and memory transfers, we compute the additional cost when processing large feature maps layer- and depth-wise. For large frames, the cost is clearly dominated by the external memory access energy. Table 3.4 shows an exploration over different frame sizes starting from 32×32 for which no tiling is required and extending to 64×64 and 96×96 that require significant external memory transfer. We find that by minimizing the feature map movement, the depth-first strategy consumes significantly less than the layer-first strategy for practical cases.

While the CUTIE core is designed to be integrated with a host processor, one key idea to reduce system-level energy consumption realized in the architecture is the autonomous operation of the accelerator core. The control implementation allows the accelerator to compute a complete network without interaction with the host. In the presented

version, the weight memories, the layer FIFO, and threshold FIFOs are designed to store up to eight full layers, which can be scheduled one after another without any further input. In general, the number of layers can be freely configured, at the cost of additional FIFO and weight memory.

Besides offering support for standard convolutional layers, the architecture can be used for depthwise convolutional layers by using weight kernels where each kernel is all zeros except for one channel. Further, it can be used for ternary dense layers with input size smaller or equal to $3 \times 3 \times 128 = 1'152$ and output size smaller or equal to 128 by mapping all dense layer matrix weights to the $3 \times 3 \times 128$ weight buffer of an OCU.

3.4 Implementation

This Section discusses the implementation of the CUTIE accelerator architecture. The results from physical layouts in a 22 nm technology, one using SCMs and another using SRAMs, and from synthesis in a 7 nm technology are presented and discussed.

3.4.1 Interface Design

The interface of the accelerator consists of a layer instruction queue and read/write interfaces to the feature map and weight memories. The interface is designed to allow integration into a SoC design targeting near-sensor processing. In this context, a pre-processing module could be connected to a sensor interface, with a host processor only managing the initial setup and off-chip communication. This setup consists of writing the weights into their respective weight memories and pre-loading the layer instructions into the instruction queue. In the actual execution phase, i.e. once data is loaded continuously, the accelerator is designed to autonomously execute the layer instructions without needing any further input besides the input feature maps and return only a highly-compressed feature map or even final labels. The end of computation is signalled by a single-bit interrupt to the host.

Table 3.2: Estimated energy consumption of a network consisting of 8 convolutional layer without pooling for tiled computation of large feature maps on a GF 22 SCM implementation including I/O and external DRAM

	Depth-first	**Layer-first**
32×32	7.3 µJ	7.3 µJ
Bit accesses from or to external memory	209 kB	209 kB
Feature map transfer energy	4.2 µJ	4.2 µJ
Weight memory transfer energy	0.3 µJ	0.3 µJ
Computational energy	2.8 µJ	2.8 µJ
64×64	277 µJ	1069 µJ
Bits moved from or to external memory	12.6 MB	52.8 MB
Feature map transfer energy	252 µJ	1057 µJ
Weight memory transfer energy	2.5 µJ	0.3 µJ
Computational energy	22.5 µJ	11.5 µJ
96×96	3734.5 µJ	6030.3 µJ
Bit accesses from or to external memory	179.3 MB	300.1 MB
Feature map transfer energy	3586 µJ	6002 µJ
Weight memory transfer energy	14.5 µJ	0.3 µJ
Computational energy	134 µJ	28 µJ

3.4.2 Dimensioning

The CUTIE architecture is not architecturally constrained to support a certain number of input/output channels, i.e. it can be parameterized to support an arbitrary amount of channels. Since it can be synthesized with support for any number of channels and feature map sizes, the proposed implementation was designed to optimize the accuracy vs. energy efficiency trade-off for the CIFAR-10 dataset. To this end, the compute units were synthesized and routed for different channel numbers to evaluate the impact of channel number on the energy efficiency of individual compute units and by extension, the whole accelerator. The estimations were performed for 64, 128, 256, and 512 channels. To estimate the energy efficiency of the individual implementations, a post-layout power simulation was performed, using randomly generated activations and weights. This experiment was repeated and averaged over 300 cycles, i.e. 300 independently randomly generated weight tensors and feature maps were used. Further, post-synthesis simulation estimations for the energy cost of memory accesses, encoding & decoding, and the buffering of activations and weights were added. The estimations for the resulting accelerator-level energy efficiency are shown in Figure 3.6. Since these estimations were made using a post-layout power simulation of a single OCU, they take into account the wiring overheads introduced by following the completely unrolled compute architecture. One of the main drivers for lower efficiency in the designs with more channels is the decrease in layout density and an increase in wiring overheads. While energy efficiency per operation does not directly imply energy per inference, it is a strong indicator of system-level efficiency.

3.4.3 Implementation Metrics

The accelerator design was implemented with a full backend flow in GlobalFoundries 22 nm FDX and synthesized in TSMC 7 nm technology. The first of two implementations based on GlobalFoundries 22 nm FDX was synthesized using SRAMs supplied with 0.8 V for feature map and weight memories and 8 track standard cells operating at 0.65 V. The second of the GF 22 nm implementations uses SCM-based feature map and weight memories as well as 8 track standard

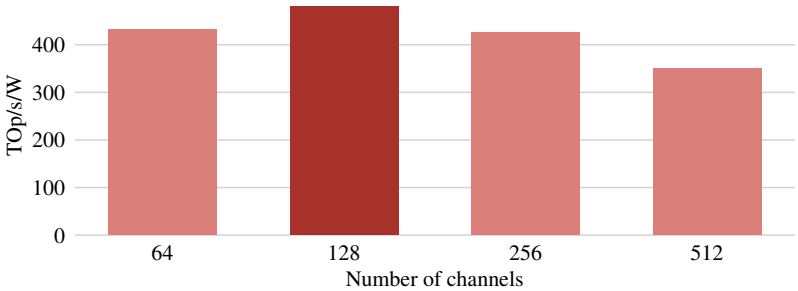

Figure 3.6: Estimation of accelerator-level energy efficiency using data from the simulation of single OCUs, assuming SCM-based memories. Feature maps and weights were drawn from a uniform random distributions. There is a peak in energy efficiency at 128 channels before falling off for increasing channel numbers.

cells for its logic cells, all supplied with 0.65 V. The TSMC 7 nm implementation similarly uses SCM-based memories to allow for voltage scaling. The post-synthesis timing reports show that the GF 22 nm implementations should be able to operate at up to 250 MHz. We chose to run both the SCM as well as the SRAM implementation at a very conservative frequency of 66 MHz. Since we did not run a full backend implementation of the 7 nm version, we chose to estimate the performance at the same clock frequency and voltage as the 22 nm versions. The total area required by the design is 7.5 mm^2 for both 22 nm implementations and approximately 1.2 mm^2 at a layout density of 0.75 for the 7 nm implementation. The reason for both GF 22 nm implementations requiring the same amount of area is due to the larger memories supported in the SRAM implementation, as explained in Section 3.3.5. A breakdown of the area usage in the SCM-based 22 nm implementation is shown in Figure 3.7.

For the GF 22 nm implementations, the sequential and memory cells take up around 80% of the overall design's area, while the clock buffers and inverters constitute only a very small amount of the total area. This characteristic is due to the choice of using latch-based buffers for a lot of the design and clocking the accelerator at a comparatively low frequency, while also extensively making use of clock-gating at every

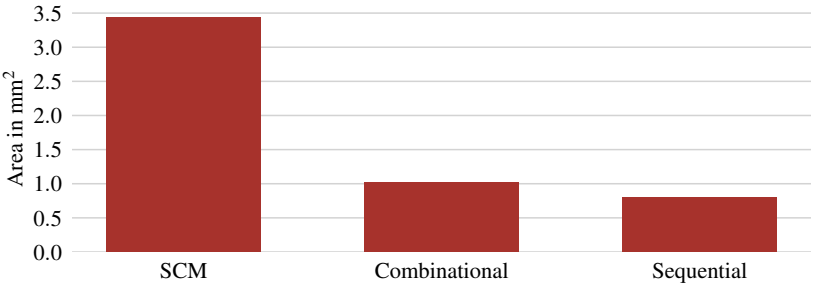

Figure 3.7: Breakdown of the area usage of the SCM implementation of the accelerator core in 22 nm technology. The majority of the area is used by the standard cell memories, which are used to store feature maps and weight kernels. Clock area is negligibly small, due to deliberate low clock speeds and hierarchical clock gating

level of the design's hierarchy. Note that even though the area of the design is storage-dominated, power and energy are not, which is one of the key reasons for the extreme energy efficiency of CUTIE.

3.5 Results and Discussion

This Section discusses the evaluation results of the proposed accelerator design. First, we discuss the design and training of the network that is used to evaluate the accelerator's performance. Next, we discuss the general evaluation setup. Finally, we present the implementation and performance metrics and compare our design to previous work.

3.5.1 Quantized Network Training

The accelerator was evaluated using a binarized and a ternarized version of a neural network, using the binary thermometer encoding and the ternary thermometer encoding for input encoding. The network architecture is shown in Table 3.3.

Each convolutional layer is followed by a batch normalization layer and a Hardtanh activation [195] layer. For the quantized versions

Table 3.3: Layer architecture of the tested CNN

Layer	Input Dim	Op	Kernel	Padding
2D Convolution	126×32×32	297 MOp	3×3	(1,1)
2D Convolution	128×32×32	302 MOp	3×3	(1,1)
2D Convolution	128×32×32	302 MOp	3×3	(1,1)
Max Pooling	128×32×32	-	2×2	(0,0)
2D Convolution	128×16×16	75.5 MOp	3×3	(1,1)
2D Convolution	128×16×16	75.5 MOp	3×3	(1,1)
Max Pooling	128×16×16	-	2×2	(0,0)
2D Convolution	128×8×8	18.9 MOp	3×3	(1,1)
2D Convolution	128×8×8	18.9 MOp	3×3	(1,1)
Max Pooling	128×8×8	-	2×2	(0,0)
2D Convolution	128×4×4	4.7 MOp	3×3	(1,1)
Avg Pooling	128×4×4	-	4×4	(0,0)
Fully connected	128	2.6 KOp	-	-
Total	-	**1.1 GOp**	-	-

of the network, the activation layer is followed by a ternarization layer. The preceding convolutional layer, batch normalization layer and Hardtanh activation layer are merged into a single Fused Convolution layer. Any succeeding pooling layers are then merged as well. The reason for using Hardtanh activations over, for example, the more popular ReLU activation which is also usually used in BNNs is the inclusion of all three ternary values in the range of the function. We further found that the Hardtanh activation converged much more reliably than the ReLU activation for the experiments we ran. We have tested networks with depthwise-separable convolutions in place of standard convolutions but have found that accuracy decreases substantially when ternarizing these networks, which is in line with the results in [167]. Further, depthwise-separable convolutions require twice the feature map data movement, while performing fewer operations overall. Since CUTIE's architecture greatly reduces the cost of the elementary multiply and add operations, the cost of accessing local buffers is relatively high. Hence, layers that have been optimized in a traditional setting to minimize the number of operations are not guaranteed to be energy efficient.

The approach for training the networks taken in this work is based on the INQ algorithm [163]. Training is done in full-precision for a certain number of epochs, after which a pre-defined ratio of all weights are quantized according to a quantization schedule. These two steps are iterated until all weights are quantized. One degree of freedom in this algorithm is the order in which the weights are quantized, called the quantization strategy. We evaluated three quantization strategies for their impact on accuracy, and sparsity, which is linked to energy efficiency for execution on the proposed architecture. The strategies evaluated in this work are the following:

- Magnitude: Weights are sorted in descending order by their absolute value
- Magnitude-Inverse: Weights are sorted in ascending order by their absolute value
- Zig-Zag: Weights are sorted by taking the remaining smallest and largest values one after another.

For both the ternarized and binarized versions, the weights were quantized using the quantization schedule shown in Figure 3.8. The CIFAR-10 dataset was used for training and the CIFAR-10 test data set was used for all evaluations. The network was trained using the ADAM optimizer [106] over a total of 200 epochs.

3.5.2 Evaluation Setup

In addition to the quantized network, a testbench was implemented to simulate the cycle-accurate behavior of the accelerator core. The testbench generates all necessary signals to load all weights and feature maps into the accelerator core and load the layer instructions into the layer FIFO. The 22 nm implementations were simulated using annotated switching activities from their respective post-layout netlist to simulate the average power consumption of the accelerator core, including memories, during the execution of each layer. Analogously, the 7 nm implementation was simulated using its post-synthesis netlist. For power simulation purposes, each layer was run separately from the rest of the network. This guarantees that each loading phase is associated with its layer, which is required to properly

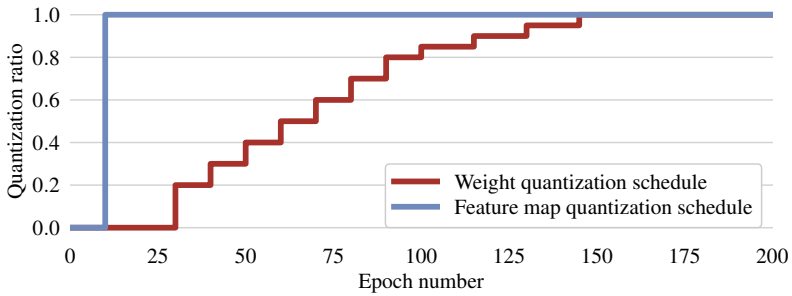

Figure 3.8: Quantization schedule for the presented network. Weights and feature map pixels are quantized separately, using different schedules. The weight quantization schedule uses a decaying step size, which starts at 20%, decreases to 10% and finishes with 5% of all weights.

estimate the energy consumption of a layer. For throughput and efficiency calculations, the following formula for the number of operations in convolutional layers is used:

$$\Gamma = 2 \cdot I_W \cdot I_H \cdot K \cdot K \cdot N_I \cdot N_O$$

where K corresponds to the side length of the convolutional kernel, I_W and I_H are the output features maps' width and height, and N_I & N_O are the input and output channel number, respectively. Γ corresponds to the number of additions and multiplications required to compute each output pixel, i.e. operations for pooling and activations are not considered. Furthermore, the runtime of each layer is measured between the loading of the layer instruction and the write operation for the last output feature map pixel.

3.5.3 Experimental Results

The energy per operation for the 22 nm implementation using different quantization strategies is shown in Figure 3.11. The energy efficiency scales almost linearly with the sparsity of the executed network. This

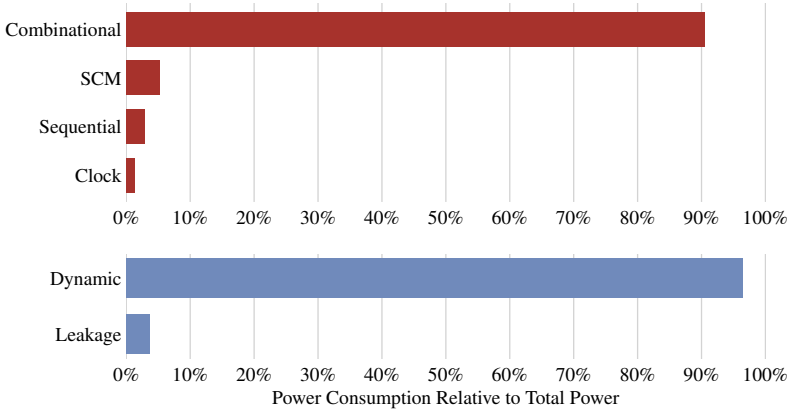

Figure 3.9: Power breakdown of the accelerator core implementation in 22 nm technology with SCM-based feature map and weight memories, running the Magnitude-Inverse trained ternary network. The overall power is clearly dominated by combinational cells, where over 90% of the total power is spent.

trend can be explained by zeros in the adder trees leading to nodes not toggling, which results in lower overall activity.

A breakdown of power consumption by cell type, as well as by dynamic and leakage power is shown in Figure 3.9. The static power consumption makes up 4.6% of the overall power consumption in the 22 nm implementation, most of which stems from the SCMs. Notably, the power consumption is dominated by combinational cells which underlines the effectiveness of the architecture, since this implies most energy is spent in computations, rather than memory accesses or transfers.

The analysis of the per-layer energy efficiency for both binary and ternary neural networks reveals a sharp peak in the first layer, which can be explained with the structural properties of the thermometer encoding, i.e. the first feature map contains 66.3% zeros on average. Furthermore, with the decreasing number of operations in deeper layers, the energy cost of loading the weights increase in proportion to

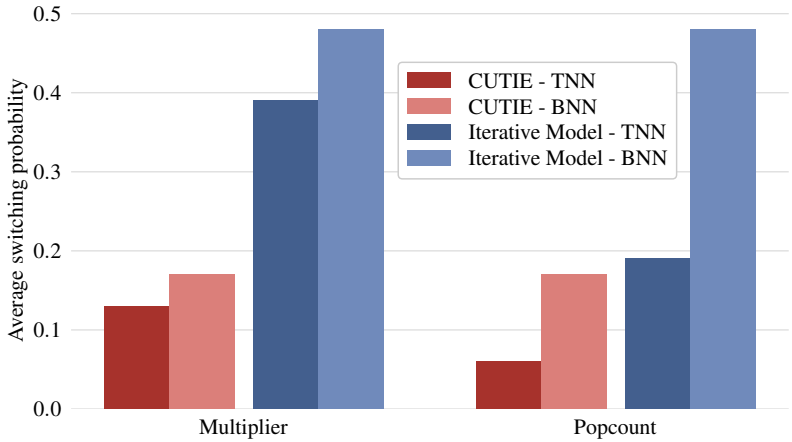

Figure 3.10: Overview of the switching probabilities at the multiplier and adder tree input nodes respectively, smaller is better. For the binary case, toggling in the multipliers directly translates to switching activity in the adder trees, while for the ternary case the sparsity of the network reduces switching activity at the adder tree input nodes by $\approx 2\times$. Moreover, the smoothness of feature maps is exploited by unrolling the compute units, which is reflected in a $\approx 3\times$ smaller switching probability compared to an iteratively decomposed model. Best viewed in color.

the energy cost of computations, which explains the decreasing energy efficiency in deeper layers.

The binary thermometer encoding and ternary thermometer encoding were compared for their use with the ternarized network version. The results show that the ternary thermometer encoding provides a small increase between 0.5% and 1.5% in test accuracy, while energy efficiency is kept within 2% of the binary thermometer. Further, the drop in accuracy between the 32-bit full-precision version and the ternary version can be reduced to as little as 3%.

Finally, the ternary network trained with the Magnitude-Inverse quantization strategy using the ternary thermometer encoding was evalu-

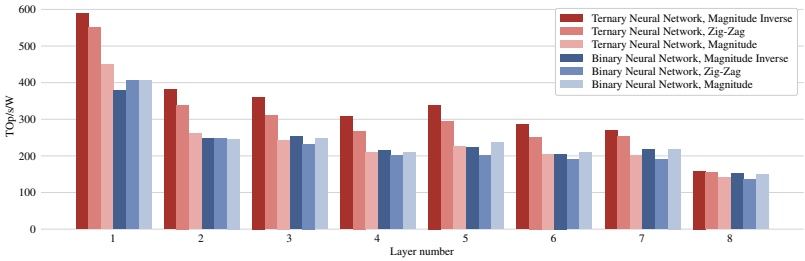

Figure 3.11: Energy efficiency simulation results on the CIFAR-10 test dataset for the binarized & ternarized networks comparing the different quantization strategies using the GF 22 nm post-layout power simulation data. Notably, the energy efficiency per operation increases with increasing sparsity of the weight kernels as shown in table 3.4.

ated on the post-synthesis netlist of the 7 nm implementation, achieving a peak energy efficiency of 3140 TOp/s/W in the first layer and an average efficiency of 2100 TOp/s/W.

3.5.4 Comparison of Quantization Strategies

An overview of test accuracy and sparsity for all tested strategies is given for the binarized and ternarized versions in Table 3.4.

The energy per inference for the most efficient ternary version in 22 nm adds up to 2.8 µJ, the energy per inference for the best binary version to about 4.4 µJ. These results allow three observations: first, the quantization strategy not only impacts the accuracy of the resulting network but also the distribution of weights - the number of zeros for the Magnitude-Inverse strategy is more than 8x higher than for Magnitude, at comparable accuracy. The second observation is that energy efficiency increases significantly for very sparse networks. The Magnitude-Inverse strategy trains a network that runs 36% more efficiently than the one trained with Magnitude for the ternary case. Lastly, the results imply that the optimal quantization strategy might be different for the binary and ternary case. Most importantly, for all

Table 3.4: Impact of quantization strategy on test accuracy and sparsity for binarized & ternarized networks on the CIFAR-10 dataset evaluated in the 22 nm SCM implementation

	Accuracy	Weighty Sparsity	Avg. TOp/s/W
Full-Precision	91%	-	-
Ternary, TT*			
Magnitude	86.5%	7.4%	260 TOp/s/W
Magnitude-Inverse	87.4%	60.7%	392 TOp/s/W
Zig-Zag	88.1%	49.1%	345 TOp/s/W
Ternary, BT*			
Magnitude	85.9%	6.9%	262 TOp/s/W
Magnitude-Inverse	86.8%	60.8%	399 TOp/s/W
Zig-Zag	86.6%	49.2%	342 TOp/s/W
Binary			
Magnitude	83.3%	0%	240 TOp/s/W
Magnitude-Inverse	80.1%	0%	248 TOp/s/W
Zig-Zag	82.8%	0%	229 TOp/s/W

* BT: Binary Thermometer
* TT: Ternary Thermometer

training experiments we have run, we have found that ternary neural networks consistently outperform their binary counterparts on the CUTIE architecture by a considerable margin, both in terms of accuracy, with 5% higher test accuracy, as well as in terms of energy efficiency, with 36% lower energy per inference.

3.5.5 Exploiting Feature Map Smoothness

By fully unrolling the compute units with respect to the feature map channels and weights, we reduce switching activity in the adder tree of the compute units by an average of 66.6% with respect to architectures that use an output-stationary approach and iterative decomposition. Iteratively decomposed architectures require the accelerator to compute partial results on partial feature maps and weight kernels. The typical approach to implement this is tiling the feature map and weight kernels in the input channel direction, and switch the weight and feature map tiles every cycle. This leads to much higher switching activity.

In the ternary case, an input node of the adder tree switches when the corresponding weight value is non-zero and the feature map value changes. Calculating the mean number of value switches between neighboring pixels, we found that the binary feature map pixels have an average Hamming distance of 44 out of 256 bit and the ternary feature map pixels have an average pixel-to-pixel Hamming distance of 33 out of 256 bit following the 3-ary encoding of CUTIE. It exploits this fact by keeping the weights fixed for the execution of a full layer, which eliminates switching activity due to changing the weight tile while a previous feature map tile is scheduled. To quantify this effect, we analyzed the switching activity of the presented network trained with all quantization strategies on an output-stationary iterative architecture model, taking into account the network weights as well. Figure 3.10 shows the occurring switching activity for CUTIE versus a model with 2× iterative decomposition for the binary Magnitude and ternary Magnitude-Inverse trained networks.

3.5.6 Comparison of Binary and Ternary Neural Networks

Since the set of ternary values includes the set of binary values, a superficial comparison between binary and ternary neural networks on the proposed accelerator architecture is fairly straight-forward, as binary neural networks can be run on the accelerator as-is. To fairly compare, however, it is important to discount certain contributions that only appear because the accelerator core supports ternary operations. Most importantly, the overhead in memory storage, accesses, encoding, and decoding should be subtracted, as well as the energy spent in the second popcount module. To apply these considerations on the architecture, the following simplifications are made:

- The power used for memory accesses is divided by 1.6.
- The power used in the popcounts of the compute units is halved.
- The power used for encoding and decoding is subtracted.

While these reductions do not account for all differences between the ternary and a binary implementation of the accelerator, they give a reasonably close estimate, considering that the power spent in popcounts, memories and encoding & decoding modules accounts for around 80% of the total power budget. Adding up the reductions, an average of around 30% should be subtracted from the measured values of the GF 22 nm SCM implementation to get an estimate for the energy efficiency of a purely binary version of the accelerator. Even including this discount factor into all calculations, the energy of the binary neural network would be reduced to around 3 µJ, which is slightly higher than the ternary version. Taking into account that the achieved accuracy for the ternary neural network comes in at around 88% while the binary version achieves around 83%, the ternary implementation is both more energy-efficient and more accurate in terms of test accuracy than the binary version.

3.5.7 Comparison with the State-of-the-Art

A comparison of our design with similar accelerators cores is shown in Table 3.5. The implementation in TSMC 7 nm technology outper-

Table 3.5: Comparison of the proposed architecture to state-of-the-art accelerators

	[21]	[155]	[26]	[159]	[156]	**This work**		
Computation Method	digital	digital	mixed	digital	analog	digital	digital	digital
Weight Precision	binary	binary	binary	binary	ternary	ternary	ternary	ternary
Activation Precision	binary	binary	binary	binary	ternary	ternary	ternary	ternary
Memory Implementation	SCM	SRAM	SRAM	SCM	SRAM	SRAM	SCM	SCM
Technology	22 nm	28 nm	28 nm	10 nm	32 nm	22 nm	22 nm	7 nm
Core Area [mm^2]	0.7	1.4	5.76	**0.39**	1.96	7.5	7.5	1.2[b]
Core Voltage [V]	0.4	0.66	0.6	**0.37**	-	0.65	0.65	0.65
Peak Throughput [TOp/s]	0.3	2.8	-	**160**	114	16	16	16
Peak Core Energy Efficiency [TOp/s/W]	223	230	-	617	-	457	589	**3'140**
Average Core Energy Efficiency [TOp/s/W]	36	145	772	617	127	305	392	**2'100**
Accuracy on CIFAR-10	87%	86%	85.6%	86%[a]	-	**88%**	**88%**	**88%**
Energy per Inference on CIFAR-10 [µJ] (excl. I/O)	1.3–7.3	13.86	2.61	3.2	-	3.6	2.8	**0.52**

[a]: uses same network as [155] [b]: expected value at 0.75 cell layout density

forms even the most efficient digital binary accelerator design, implemented in comparable Intel 10 nm technology as reported by Knag et al. [159], by a factor of at least 3.4× in terms of energy efficiency per operation and 5.9× in terms of energy per inference as well as the most efficient mixed-signal design as reported by Bankman et al. [26], requiring a factor of 4.8× less energy per inference.

For a fairer comparison to other state-of-the-art accelerators, we also report post-layout simulation results in GF 22 nm technology, which similarly outperforms comparable implementations as reported in Moons et al. [155] by a factor 2.5×, both in terms of peak efficiency as well as average efficiency per operation. The more practical comparison between the energy per inference on the same data set reveals that our design outperforms all other designs by an even larger margin, i.e. by at least 4.8×, while even increasing the inference accuracy with respect to all other designs. However, our design is less efficient in terms of throughput per area compared to other state-of-the-art designs. This is a deliberate design choice, which is due to the unrolled architecture of CUTIE.

3.6 Conclusion

In this work, we have presented three key ideas to increase the core efficiency of ultra-low bit-width neural network accelerators and evaluated their impact in terms of energy per operation by combining them in an accelerator architecture called CUTIE. The key ideas are: 1) completely unrolling the data path with respect to all feature map and filter dimensions to reduce data transfer cost and switching activity by making use of spatial feature map smoothness, 2) moving the focus from binary neural networks to ternary neural networks to capitalize on the inherent sparsity and 3) tuning training methods to increase sparsity in neural networks at iso-accuracy. Their combined effect boosts the core efficiency of digital binary and ternary accelerator architectures and contribute to what is to the best of our knowledge the first digital accelerator to surpass POp/s/W energy efficiency for neural network inference.

Chapter 4

TCN Extensions for CUTIE

In this Chapter, we introduce a novel processing scheme alongside minor hardware modifications for TCN networks on the CUTIE accelerator introduced in Chapter 3. The design achieves 5.5 µJ/Inference, 12.2 mW, 8000 Inference/s at 0.5 V for a Dynamic Vision Sensor (DVS) based TCN, and an accuracy of 94.5 % and 2.72 µJ/Inference, 12.2 mW, 3200 Inference/s at 0.5 V for a non-trivial 9-layer, 96 channels-per-layer convolutional network with CIFAR-10 accuracy of 86 %. The peak energy efficiency is 1 POp/J outperforming the state-of-the-art silicon-proven TinyML quantized accelerators by 1.67 × while achieving competitive accuracy.

The following sections have been published in a slightly different form in the IEEE Micro journal[1].

[1]

4.1 Introduction

Advances in ML research in recent years have enabled a new direction of research within the field of embedded systems, called TinyML, targeting the execution of non-trivial ML tasks within the strict constraints of low-power (mW) embedded devices. TinyML is becoming increasingly pervasive with applications including wearable computer vision, gesture recognition, and many more. The key challenges in TinyML are energy efficiency at a few mW of power while ensuring accurate and fast inference. Specialized TinyML accelerators tackle both challenges by providing high throughput at low power, but often they do so by compromising accuracy or by specializing on a single network topology, with no flexibility. We present a flexible and accurate TinyML architecture, integrating CUTIE, a highly configurable TNN accelerator based on a fully unrolled compute architecture [34] within the *Kraken* RISC-V SoC.

While CUTIE, as presented in Chapter 3, is highly efficient at processing single static images, it lacks support for processing temporal data. However, at the extreme edge, information is typically extracted from the temporal evolution of sensor data, as shown in Chapter 2. To adapt CUTIE to the requirements of extreme edge time-series processing, we introduce a novel, lightweight TCN extension within the CUTIE architecture. At their core, TCNs consist of dilated convolutions, which are calculated over the input sequence's time dimension. This allows TCN networks to model the evolution of temporal dynamics without the training and inference challenges posed by Recurrent Neural Networks (RNNs) while using several orders of magnitude fewer parameters than Transformers, making them ideal for an ultra-high-efficiency accelerator like CUTIE. In this chapter, we show how mixed CNN and TCN networks similar to TinyRadarNN described in Section 2.1 are mapped on the extended TCN-CUTIE accelerator to achieve state-of-the-art energy efficiency, by studying the implementation of a gesture recognition network trained on Dynamic Vision Sensor (DVS) data.

More specifically, we present the deployment and power measurements of the DVS neural network architecture exploiting our novel TCN extensions, which achieves an accuracy of 94.5% at an energy cost of

5.5 µJ per inference, performing gesture recognition from a DVS. To the best of our knowledge, we are the first to demonstrate a peak core energy efficiency beyond 1 PetaOp/s/W for neural network inference in an all-digital and flexible platform, measured on a fabricated SoC.

4.2 SoC Implementation

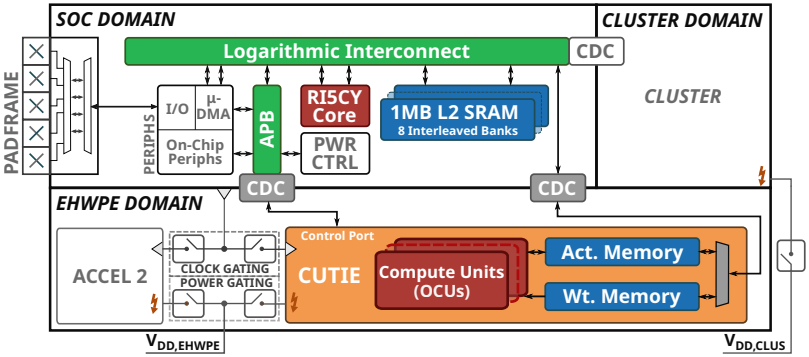

Figure 4.1: Block diagram of Kraken, including the three switchable power domains and always-on SoC domain. The CUTIE accelerator is integrated with a control port connected to the APB and a data port connected to the high-bandwidth logarithmic interconnect. The Cluster and Accel 2 IPs are not discussed in this Chapter.

The Kraken SoC is a RISC-V-based microcontroller based on the Pulpissimo SoC [196]. A RI5CY core [77] serves as the fabric controller (FC), coordinating the operation of the other subsystems. For parallel signal processing tasks, it contains an 8-core PULP cluster of RISC-V cores. Kraken has an extensive set of peripherals for off-chip communication. They are implemented as µDMA [197] extensions, freeing the FC from most management duties. On-chip peripherals include an event unit for interrupt mapping, a RISC-V-compliant debug unit for JTAG control of the chip and a power controller. 4 Frequency-Locked Loop (FLL) modules provide independently runtime configurable clocks to the µDMA peripherals, the SoC domain,

the accelerator (EHWPE) domain, and the PULP cluster. Kraken has three core supply rails and four core power domains. The SoC, cluster and EHWPE domains each have a separate supply. Kraken features task-specific accelerators: In the following sections, we focus on the TNN inference engine, its integration, and silicon measurements. The accelerators share their supply voltage, but each is located in its power domain and can be power-gated individually to minimize current draw by idle system components. A block diagram of the SoC architecture is shown in Figure 4.1.

4.3 CUTIE Design

The Completely Unrolled Ternary Inference Engine (CUTIE), is a highly configurable CNN accelerator architecture for completely ternarized neural networks, introduced in [34]. In contrast to systolic arrays, CUTIE uses a completely unrolled compute architecture, which means that one OCU is allocated for every output channel, making the computation output-stationary. Further, each OCU includes weight buffers, minimizing weight data movement. Each OCU processes a full activation window per cycle, without pipelining in the compute units, making the architecture also input-stationary. To fully exploit all opportunities for data reuse in CNNs, a linebuffer designed to eliminate data access stalling is added. Thanks to this highly parallel design, CUTIE fully exploits data reuse at all levels and minimizes data movement. In addition, ternary weights and activations enable the exploitation of zero values to translate sparsity into reduced toggling in the compute units. Thus, minimized data movement and switching activity are the cornerstones of CUTIE's efficiency.

In this Chapter, we extend the CUTIE TNN accelerator to support hybrid 2D-CNN & 1D-TCN networks. As demonstrated in [46], the combination of low precision, i.e. ternarized, CNN and TCN achieved superior accuracy in classifying time-distributed data like streams of events produced by event-based sensors, like DVS cameras. Data produced by such sensors are characterized by a high level of unstructured sparsity and exhibit both short and long temporal dynamics. A

hybrid CNN-TCN approach allows fine-tuning the network capabilities to achieve the highest accuracy when processing event streams. Specifically, the CNN captures the spatial dependency among neighboring events, that cluster in specific regions of the input feature map, as well as short temporal dependency among events belonging to consecutive time steps; an event happening in the scene tends to persist over multiple time steps. The 1D TCN extracts longer temporal dependencies among features distributed across the entire sample time window. 1D-TCNs use dilated convolutions [98], meaning feature map data is accessed in a strided fashion. The extensions required to support 1D-TCNs efficiently are twofold: 1) We designed a TCN memory, enabling dilated feature map data access without stalling, 2) We implemented a scheduling algorithm that maps 1D dilated convolutions to 2D undilated convolutions, which make use of CUTIE's efficient compute architecture.

4.4 TCN Extensions

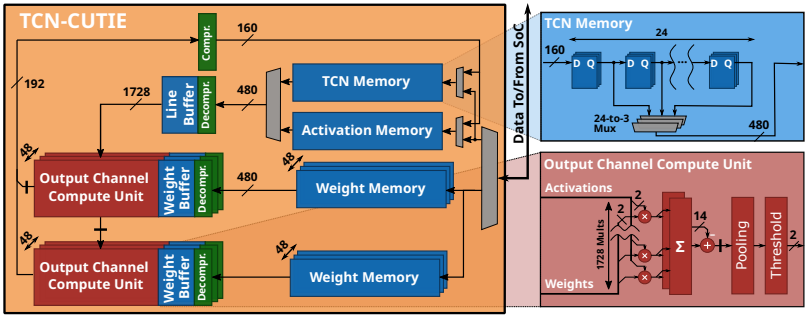

Figure 4.2: Block diagram of the 96-channel CUTIE implementation with TCN extensions, showing the completely unrolled data path. The insets show the flip-flop based TCN memory and the OCU, which processes an entire convolution window per cycle. Notably, the OCU uses a single pipeline stage.

To support hybrid 2D-CNN & 1D-TCN networks, CUTIE has to be extended with a small memory, the TCN memory, that can hold the 1D feature vectors that are extracted by each inference of a 2D-CNN. The TCN memory enables the execution of hybrid 2D-CNN & 1D-TCN networks, as well as pure 1D processing. The output of the TCN memory has the same size as the activation memory, which is achieved by multiplexing three time steps according to the address of the first required pixel. In the Kraken SoC, the TCN memory was dimensioned to hold a total of 24 feature vectors, corresponding to a memory size of only 576 bytes. Nevertheless, 24 time steps are sufficient to cover a long receptive window even at high framerates: if the 2D CNN takes as input 15 stacked frames captured at a rate of 300 FPS ($5-10\times$ the speed of most ordinary cameras), the resulting receptive time window for a TCN covering 24 time steps is still 1.2 s. Due to its small size, we implemented the TCN memory/ as a flip-flop-based shift register to reduce leakage power. A block diagram of the CUTIE TNN accelerator with the proposed TCN memory extension is shown in Figure 4.2.

The second extension we introduce to the CUTIE accelerator is the mapping of 1D dilated convolutions. Dilated 1D convolutions with a kernel length N and dilation factor D of an input x with a kernel w can be described by their mathematical definition, shown in Equation 4.1:

$$(w \star x)[n] = \sum_{k=1}^{N} \tilde{x}[n-(k-1)\cdot D] \cdot w[N-k] \tag{4.1}$$

where

$$\tilde{x}[n] = \begin{cases} x[n], & n \geq 0 \\ 0, & else \end{cases}$$

is the *causally padded* input vector x. The main advantage of dilated convolutions over undilated ones lies in their ability to reach a longer receptive field in fewer layers. In a TCN with $N = 3$ and $D_i = 2^i$, where D_i denotes the i-th layer's kernel dilation, the receptive field f_k in layer k can be calculated as

$$f_k = 1 + \sum_{i=0}^{k} (N-1) \times 2^i$$

. The receptive field increases exponentially with the number of layers, decreasing the number of layers needed to cover a given number of input steps. For the 24 input steps supported by TCN-CUTIE, the number of layers is reduced from 12 for undilated convolutions to 5 with exponentially increasing dilations. In a direct implementation, the elements of $\tilde{x}$ are not accessed contiguously, instead, they are accessed with a stride of D. Due to the specialized memory hierarchy of CUTIE, non-contiguous or strided accesses lead to stalling, decreasing efficiency. To avoid this, we reformulate equation 4.1 as a 2D correlation:

$$(w \star x)[n] = \sum_{k=1}^{N} z[N-k, mod(n, D)] \cdot w[N-k]$$

where

$$z[n, m] = \tilde{x}[n \cdot D + m]$$

A visual representation of this mapping is shown in Figure 4.3. To form the dense 2D feature map, the 1D vector is wrapped around after D elements. Further, zero padding (shown in white in Figure 4.3) is applied on the edges to implement the causality required by TCNs as well as the correct start- and endpoint of the convolution. To respect the hardware constraints of CUTIE, i.e. weight kernels having size 3×3, the 1D weight kernel is projected into the middle column of the 2D weight kernel, while all other elements in the weight kernels are set to zero. This mapping ensures that the kernel dot product is only computed over a single column and the column elements are dilated by the dilation factor D. Since this mapping is fully equivalent to a 2D convolutional layer and all transforms necessary can be computed offline and require no data marshalling, it fully and efficiently reuses the CUTIE architecture with minor hardware overhead.

4.5 TCN-CUTIE Implementation

Thanks to its highly configurable nature, the CUTIE architecture can be adapted to many application scenarios. In the Kraken SoC, we dimensioned the memories for feature map sizes of up to 64×64 pixels

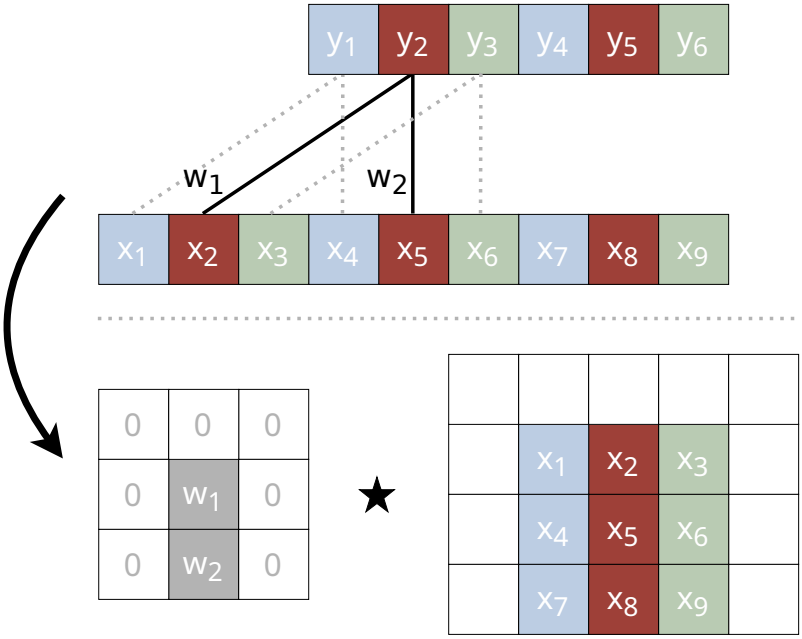

Figure 4.3: Example mapping of a dilated 1D convolution to an undilated 2D convolution for $D = 3$, $N = 2$

with up to 96 channels. We designed the TCN memory to hold a total of 24 time steps. Since CUTIE's throughput per cycle is enormous due to the high degree of parallelism, we used relaxed timing constraints during synthesis, to enable extensive instantiation of low-leakage library cells. The CUTIE TNN accelerator's clock can be hierarchically clock-gated to minimize idle switching activity in idle OCUs when network layers have a small number of output channels. Inference can be triggered via a configuration register or an interrupt line from I/O peripherals, enabling autonomous data preparation and inference without intervention from the FC. After inference has concluded, CUTIE asserts an interrupt which is used to wake up the FC.

4.6 Kraken Physical Implementation

The Kraken chip has been designed and manufactured in *GlobalFoundries* 22nm technology, the total die area is $9\,\mathrm{mm}^2$. The three Kraken subsystems are implemented as independent clock and power domains. Both the general-purpose RISC-V-based accelerator and the EHWPE domain can be entirely power-gated to reduce their leakage consumption when not in use. The chip can operate in a wide supply voltage range, i.e., from $0.5\,\mathrm{V}$ to $0.9\,\mathrm{V}$. The chip host a total of 88 pads, 46 of which can be used either as GPIO or as an alternate function, i.e., as one of the signals of each IO peripherals, in an all-to-all muxing scheme. Figure 4.4 shows an annotated floorplan of the Kraken SoC, including the SoC Domain, Cluster, Accelerator 2, as well as CUTIE. The CUTIE accelerator occupies $2.96\,\mathrm{mm}^2$ of area. In the CUTIE layout, the area occupied by memory macros composing the internal buffers, and digital logic are highlighted. The memories including weight buffers in the OCUs take up 60% of the total die area of CUTIE, while the rest is used by the compute units. The additional TCN memory which holds the sequence samples was implemented in SCM and has a negligible impact of less than 1% on area.

4.7 Evaluation

To benchmark the accelerator's performance against similar state-of-the-art designs, we measure the execution of a ternarized 9-layer (8 CONV layers, 1 FC classifier) CIFAR-10 network as used in [34, 155, 159] with 96 instead of 128 channels. This network achieves an accuracy of 86% on CIFAR-10, which is on par with the binarized version using 128 layers used in [155, 159]. Similarly, we execute the hybrid 2D-CNN & 1D-TCN network proposed in [46], consisting of 5 2D-CNN layers and 4 1D-TCN layers that process 5 time steps. This network achieves an accuracy of 94.5% on the 12-class DVS 128 dataset.

To evaluate the power consumption of CUTIE, we measured the current drawn by the Kraken ASIC on an ASIC tester, while running the deployed networks on CUTIE using pre-selected inputs which were randomly drawn from the respective validation set of the datasets

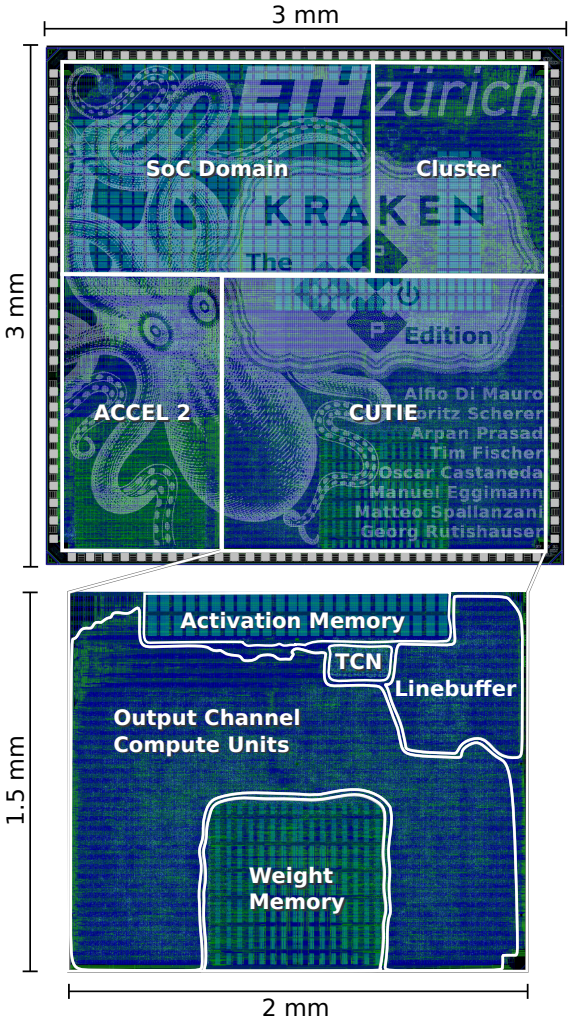

Figure 4.4: Die micrograph of the Kraken SoC. The top floorplan shows the four power domains, including SoC, Cluster, Accel 2 and CUTIE. The bottom floorplan shows the layout of modules within CUTIE.

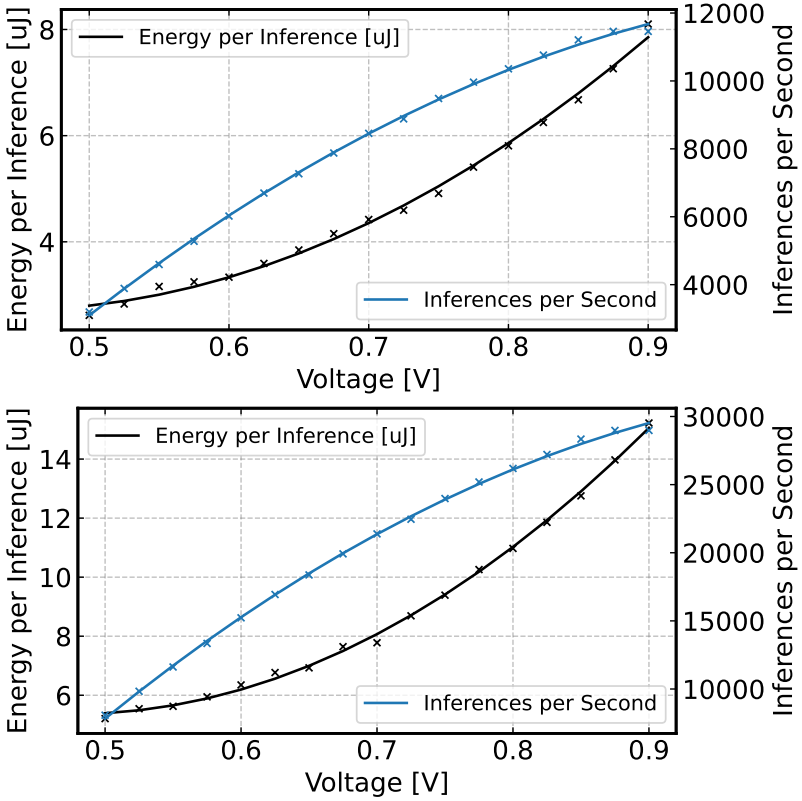

Figure 4.5: Energy per inference and inferences per second for the CIFAR-10 (upper) and DVS (lower) networks plotted against voltage using the maximum stable frequency at each corner. All data was recorded at 25°C.

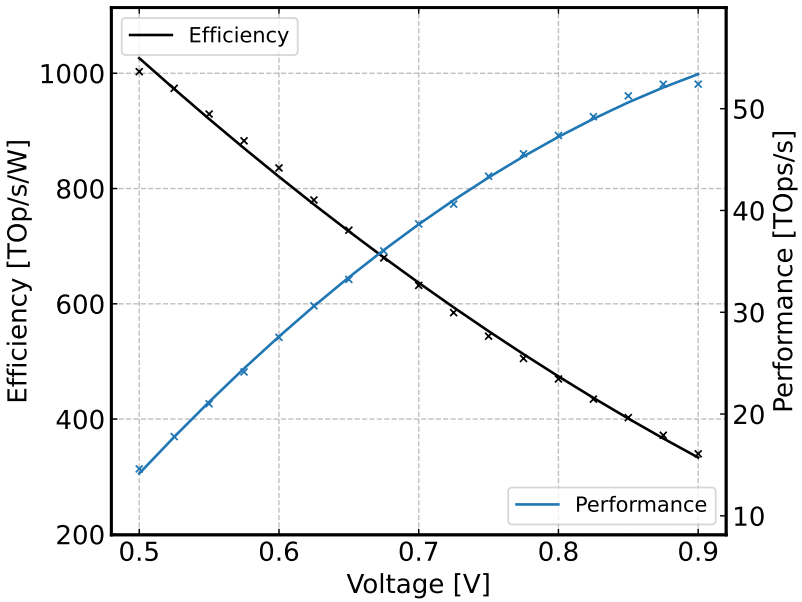

Figure 4.6: Peak energy efficiency and throughput plotted against voltage. One MAC operation corresponds to 2 Ops. Data is calculated at the maximal frequency for each voltage corner. All data was recorded at 25°C.

used for training. The presented power consumption numbers of the CUTIE accelerator include its memories, but do not include chip I/O energy. All measurements were performed at room temperature. We profiled the accelerator's performance at 25C over a range of 0.5 V - 0.9 V. Below 0.5 V, the integrated SRAM macros start exhibiting bit errors. In terms of efficiency, we find that the 0.5 V operating corner, operating at 54 MHz achieves the lowest energy per inference of 2.72 µJ and 5.5 µJ at an average throughput of 5.4 TOp/s and 1.2 TOp/s for the CIFAR-10 and DVS networks, with a peak energy efficiency in the first layer of the CIFAR-10 network of 1036 TOp/s/W and peak throughput of 14.9 TOp/s. The operating corner using 0.9 V achieves the highest peak throughput of 51.7 TOp/s, but a lower peak energy efficiency of 318 TOp/s/W. Figure 4.5 shows plots for throughput and energy per inference against voltage for the CIFAR-10 and DVS networks. Figure 4.6 shows the peak energy efficiency per operation and throughput versus voltage for the first layer of the CIFAR-10 network.

4.8 Comparison with State-of-the-Art

Table 4.1 shows a comparison of CUTIE on the Kraken SoC with state-of-the-art highly quantized digital convolutional network accelerators. CUTIE achieves a peak throughput of 56 TOp/s and a peak energy efficiency of 1036 TOp/s/W, surpassing the highest reported efficiency in the literature: a BNN accelerator manufactured in a more advanced technology node [159]. As demonstrated in [34] by the use of post-layout simulation of a larger configuration of CUTIE, the high energy efficiency of CUTIE can be explained mainly by two design characteristics: The source of CUTIE's efficiency is the minimization of data movement, which limits the efficiency of comparable accelerators. This is achieved by the fully unrolled architecture, which minimizes the number of accesses to each data item. Secondly, the simple ternary processing elements and the use of very wide addition trees leverages sparsity in ternary data indirectly. This is shown in [34], where ternarized networks with very sparse activations and weights reduce the inference energy cost on CUTIE by 36%. In this Chapter, we improve on these characteristics by optimizations in the front- and backend design flow, as well as using a smaller CUTIE configuration.

Table 4.1: Comparison of CUTIE with SoA highly quantized digital accelerators. All listed papers use the CIFAR-10 dataset and 9-layer CNN, however, this Chapter uses 96 channels instead of 128.

Characteristics	[155]	[159]		**This work**	
Computation Method	digital	digital		digital	
Weight Precision	binary	binary		ternary	
Activation Precision	binary	binary		ternary	
Technology	28 nm	10 nm		22 nm	
Dataset	CIFAR-10	CIFAR-10		CIFAR-10	
Accuracy	86%	86%		86%	
Energy per Inference	13.86 µJ	3.2 µJ		2.72 µJ	
Core Area [mm^2]	1.4	**0.39**		2.96	
Voltage [V]	0.65	0.37	0.75	0.5	0.9
Throughput [TOp/s]	2.8	3.4	**163**	16	56
Peak Core Energy Efficiency [TOp/s/W]	230	617	269	**1036**	446

To evaluate our TCN extensions we compare our design with traditional TCN accelerators, as well as with Spiking Neural Network (SNN) accelerators, which purportedly are more energy efficient for sparse, event-based time-series data like DVS.

Although there are no standard benchmarks or datasets for TCNs, we can compare the average energy efficiency over an inference for state-of-the-art designs. In [198], the authors propose a TCN accelerator design for continuous, ultra-low-power keyword spotting. While running 64 inferences of a 1.5 MOp/inference network per second, they achieve an average power consumption between 5 µW and 15 µW, leading to average energy efficiency of 6.4 TOp/s/W to 19.2 TOp/s/W, measured by post-synthesis simulation. In direct comparison, our measured average energy cost per operation on the DVS network is around 5 - 15× lower.

Even when comparing the performance of the TCN extensions with state-of-the-art SNN accelerators on the DVS 128 dataset, our implementation meets the best reported accuracy using TNNs in literature, a network deployed on the IBM Truenorth platform, which achieves a

statistical accuracy of 94.6%, just 0.1% better than our ternary TCN, while requiring 3250× more energy per inference [199] than our design.

When comparing with a modern Intel 14 nm accelerator implementation, the measured energy per inference of 5.5 µJ beats the best reported energy efficiency on a similar DVS and EMG dataset, an SNN running on the Intel Loihi platform and achieving an accuracy of 96.0%, by a factor of 63.4× [200].

4.9 Conclusion

We presented the CUTIE implementation in the Kraken SoC and evaluated its performance. By exploiting minimized data movement and switching activity coupled with aggressive voltage scaling, we achieve a peak efficiency of 1036 TOp/s/W, surpassing the SoA in ultra-low-energy CNN inference by a factor of 1.67×. Similarly, the implemented TCN extensions are demonstrated to surpass the energy efficiency of the state-of-the-art by a factor of 5×.

4.10 Outlook

While we studied applications for TCN-CUTIE using DVS camera data in this chapter, we believe that most TinyML use-case could leverage our approach of mixed CNN and TCN networks. Further research could study applications in the audio domain, or in applications using accelerometer data. Recent studies have also shown the applicability of TCNs in biomedical applications [201, 202], where mapping to TCN-CUTIE might also be explored to achieve higher energy efficiency in battery-powered devices.

Chapter 5

Deeploy: Automatic DNN Deployment for TinyML SoCs

In this chapter, we demonstrate high-efficiency end-to-end SLM deployment on a Siracusa, a multicore RISC-V (RV32) MCU augmented with ML instruction extensions and a hardware NPU, N-EUREKA. To automate the exploration of the constrained, multi-dimensional memory vs. computation tradeoffs involved in aggressive SLM deployment on heterogeneous (multicore+NPU) resources, we introduce Deeploy, a novel DNN compiler, which generates highly-optimized C code requiring minimal runtime support. We demonstrate that Deeploy generates end-to-end code for executing SLMs, fully exploiting the RV32 cores' instruction extensions and the NPU: We achieve leading-edge energy and throughput of 490 µJ/Token, at 340 Token/s for an SLM trained on the TinyStories dataset, running for the first time on an MCU-class device without external memory.

The following sections are adapted from a manuscript currently under review at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems[1].

5.1 Introduction

Following the success of Foundation Models (FMs) in NLP [203, 204], an increasing number of fields are starting to formulate and adapt FMs for high dimensional sensor data that has traditionally been challenging to process, like decoding neural data [205, 206], or training embodied AI agents [207, 208], which may incorporate multi-modal sensor inputs.

Operating directly on sensory data and in a cyber-physical loop may lead to solving many outstanding challenges in fields such as brain-machine interfaces [206] and miniaturized robotics [208]. However, to materialize this promise, models of this class need to be *embodied* in physical devices as Embodied Foundation Models (EFMs), and they must cope with the strict constraints in terms of compute throughput, power consumption, and footprint typical of edge devices. Unlike datacenter-scale systems, which collect and aggregate sensor data over sharded resources for high-throughput processing, embodied AI systems must process sensor data with extremely low latency and memory capacity under tight power constraints. This is particularly challenging for the smallest class of AI-oriented computers: so-called "*TinyML*" devices operating at the extreme edge, based on microcontroller-class devices without complex operating systems or MMUs, relying on user-level software to implement low-level hardware management functionalities. Despite many recent successes with previous-generation DNNs, the emergence of the TinyML paradigm for EFMs faces the dual challenge of reducing FMs to a manageable size and enabling their deployment on tiny devices.

[1] M. Scherer, L. Macan, V. Jung, P. Wiese, A. Burrello, F. Conti, and L. Benini, "Deeploy: Enabling Energy-Efficient Deployment of Small Language Models On Heterogeneous Microcontrollers," Mar. 2024, under Review at IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)

A first concrete step in this direction is the recent introduction of SLMs: FMs with tens to a few hundred million, rather than several billion parameters [209, 210]. While most currently available FMs are focused on processing natural language at a proof-of-concept scale, the effort towards embedded multi-modal sensor inputs with small-scale, application-specific FMs offers a highly promising path for the development of this novel class of models. Much like what happened with the initial emergence of Deep Learning [211], the evolution of advanced TinyML applications based on EFMs is currently prevented by the lack of suitable targets for deployment of these models and, even more, of deployment frameworks that enable utilizing existing specialized hardware to its full capabilities.

Deploying tiny EFMs requires overcoming several challenges specific to the TinyML domain. Large-scale AI inference systems typically employ heterogeneous computer architectures composed by a conventional host (e.g., an x86 processor) and a very large throughput-oriented accelerator (e.g., H100 [212], TPU [2]), which is fully exploited only at large batch sizes. Conversely, TinyML is used for latency-sensitive applications focusing on real-time inference without batching. As a consequence, TinyML AI inference typically employs much more specialized accelerator architectures [25, 30], leading to more complex mapping and optimization challenges for DNN deployment. Furthermore, TinyML's strict constraints on energy efficiency and microcontroller-class computer architecture typically require platform-specific optimization, including memory-aware tiling, static memory allocation, and latency-hiding Direct Memory Access (DMA) scheduling, which require advanced compiler support to scale to complex DNNs like FMs. While several compilers have limited support for user-defined kernels [213, 214], configuring and extending them requires expert knowledge, and their top-down compilation approach often clashes with loosely coupled accelerators. Moreover, mainstream compilers do not address the strict memory constraints in extreme-edge devices.

In this chapter, we aim to remove the first barrier towards developing EFM suited for deployment on TinyML platforms: the lack of deployment frameworks that enable their efficient execution. We demonstrate, to the best of our knowledge, the first end-to-end tool flow to

deploy EFMs on heterogeneous microcontroller-class systems. Specifically, we demonstrate the end-to-end deployment of a TinyStories-class [209] network on *Siracusa*, an advanced microcontroller in TSMC 16 nm technology featuring embedded non-volatile memory (MRAM) and two heterogeneous compute engines, namely, an octa-core RV32 compute cluster with instruction extensions for ML and a multi-mode CNN NPU, *N-Eureka* [30]. We present the tooling and algorithms integrated within our deployment framework, Deeploy.

The contributions of this chapter are as follows:

- We describe *Deeploy*, a customizable, domain-specific compiler designed for generating bare metal code fitting the memory constraints of extreme edge devices. Deeploy supports all the key computational primitives needed for the execution of Transformer-based EFMs on heterogeneous extreme edge SoCs through its bottom-up compilation approach, which allows applying advanced code optimization on expert-optimized kernel templates. We further introduce a novel algorithm for solving the tiling and static memory allocation problems for multi-level software-managed caches and its integration into Deeploy.
- We benchmark common Transformer encoder layer configurations, demonstrating that code generated by Deeploy maximizes engine utilization in heterogeneous, multi-accelerator SoCs. We achieve data marshaling overheads of just 9 % for large workloads with high arithmetic intensity executing on the cluster cores and NPU collaboratively thanks to efficient data movement acceleration and low-overhead offloading mechanisms.
- As a concrete large-scale end-to-end use-case of Deeploy and its adaptability to heterogeneous hardware platforms, we demonstrate for the first time the deployment of a TinyStories-class SLM on Siracusa, a state-of-the-art heterogeneous MCU. While using on-chip memory only, we achieve a throughput of 340 Token/s at an energy cost of 490 µJ for autoregressive inference. We show that using the flexible deployment flow enabled by Deeploy for the same SLM allows us to implement multi-layer KV caching using on-chip memory only, improving

token throughput by 26× compared to inference without caches.

The rest of this chapter is organized as follows: in Section 5.2, previous work in quantized neural networks, small language models, and neural network deployment for extreme edge devices is introduced and discussed. Section 5.3 introduces Deeploy and discusses its deployment flow for Transformers. Section 5.4 discusses the SLM architecture used in this Chapter and the approach to mapping it on Siracusa. In Section 5.5, we present the Siracusa MCU platform. Section 5.6 presents and discusses the end-to-end deployment results, comparing them to the state-of-the-art. Finally, Section 5.7 concludes this chapter, summarizing the results and contributions.

5.2 Related Work

This Section gives an overview of the state-of-the-art on EFMs, focusing on developments towards improvements in energy efficiency and model size and tools to deploy DNNs on extreme edge devices.

5.2.1 Small Foundation Models

Recently, the development of decoder-only Large Language Models (LLMs) such as Llama [203], and Mixtral [215], and their associated ML pipelines led to a new model type: the Foundation Model (FM).

FMs are pre-trained LLMs, which can be fine-tuned for downstream tasks at a fraction of the cost of pre-training, making them particularly relevant for domain specialization. These models have shown remarkable success in generating coherent and contextually relevant outputs, much above the capabilities of more traditional architecture like RNNs. However, LLMs often contain several billion parameters, requiring GiB of storage space, making them incompatible with extreme edge inference.

Addressing this gap, the emerging field of SLMs has gained significant traction in the last year. The aim of SLMs is to compact LLMs down to tens to hundreds of MiB [209, 210], mirroring the evolution of compression of CNNs [15] over the past decade.

This paradigm shift towards compact FMs is particularly interesting for TinyML applications. Incorporating smaller FMs, like SLMs, into embedded devices may enable a new wave of intelligent, responsive, and autonomous devices built on EFMs. Such systems could bridge the gap between human-understandable inputs such as text and performing high-level planning and low-level control tasks [216] and make such advanced capabilities available at the edge, embodied in robots, appliances, and wearable devices.

In this Chapter, we contribute to the growing field of SLM and EFM research and aim to lay the foundation for truly embedded SLMs by providing a foundational deployment flow that supports a wide range of FMs, from autoregressive decoder-only ones to encoder-only ones.

5.2.2 Quantized Transformer Models

Neural network quantization has been an active field of research for the past decade, as the promises of reduced parameter storage and higher compute efficiency on reduced-precision operands drive the development of increasingly aggressive quantization methods [13, 15].

Improvements in energy efficiency are significant when switching from floating-point point computation to integer arithmetic [17, 128] due to the reduced hardware complexity required to implement the fundamental operations using integer arithmetic. One commonly used approach to quantize DNNs is QAT, where the model is trained to overcome quantization effects that occur when using lower-precision values for weights and activations [16, 18]. However, QAT often requires computationally expensive retraining of the model and access to representative datasets, which are not readily available. PTQ methods can be applied to quantize models without retraining while conserving full-precision accuracy [217, 218]. Especially in the domain of FMs, PTQ has been successfully applied [219, 220] to reduce the computational cost of quantization. In this Chapter, we apply state-of-the-art PTQ on a publicly available pretrained SLM to achieve quantized inference without loss of accuracy, a prerequisite for energy-efficient inference on extreme edge devices.

5.2.3 Neural Network Deployment for Extreme Edge Devices

Building on the trends of model quantization and compression, as well as research into more computationally efficient DNNs [221], DNN inference on mobile and embedded devices has become a flourishing field of research [25, 30, 222]. While model deployment on mobile devices like smartphones follows similar approaches to server-scale deployment, relying on the ample compute- and memory resources, hardware-managed caches, and operating systems to carry out task scheduling available to this class of devices, deeply embedded devices face much more severe constraints in deployment. This is especially true for the new generation of MCU-class devices focusing on AI applications. In contrast to their predecessors, these MCUs feature multi-core compute clusters, DNN accelerators, and on-chip memory of up to 10 MiB, split into multiple software-managed memory hierarchy levels [25, 28, 30].

To optimally leverage the compute capabilities of such complex systems, network deployment must simultaneously optimize the execution schedule and tiling of operators and orchestrate overlapping memory transfers using DMAs to achieve low data marshaling overheads and high compute utilization. While modern top-down compilers like MLIR and TVM [213,214] allow integration of most common ISAs and accelerator APIs, their focus is not on meeting the stringent memory constraints of this class of TinyML devices. Prior work like Dory [3], CoSa [31], and others [223, 224] have addressed these challenges for CNNs by focusing on operator tiling to fit the target's memory constraints. However, these approaches assume a single-cluster memory hierarchy, with undivided memory at each level, and a simple lifetime model for network tensors, which are fundamentally stateless across inference rounds. These simplifying assumptions do not hold for complex heterogeneous multi-accelerator hardware and advanced SLM networks [225, 226].

Moving beyond these prior works, we propose a novel constraint programming algorithm that enables co-optimizing tiling and memory allocation, which overcomes the limitations of previous approaches by

supporting data flows with complex lifetimes (e.g. KV caching) as required by EFMs.

5.3 Deeploy

In this Section, we provide an overview of the Deeploy compilation flow. In contrast to most state-of-the-art compilers for DNNs, which lower DNN representations top-down into predefined primitives that need to be implemented by each backend [8,213,214], Deeploy employs a bottom-up compilation approach, where the compiler implements networks by composing user-provided C kernels, extending them with code generation passes to implement tiling and memory allocation. This bottom-up approach to compilation provides three key advantages: first, it supports reusing hand-optimized kernel libraries commonly available for most ISAs and accelerators. Second, it can be easily extended to support highly customized non-standard compute platforms, including heterogeneous SoCs featuring multiple accelerators for which a low-level compiler backend may not exist. Third, it allows easy integration of novel operators found in emerging Transformer architectures without invasive modifications to the deployment flow.

Deeploy is organized in three building blocks; the *Frontend* validates and transforms the graph representation into a representation that suits the platform and assigns kernel templates to each operator. The *Midend* performs all tiling and static memory allocation computations, guaranteeing that the computed program schedule may execute without unscheduled runtime memory spills. Finally, the *Backend* uses the optimized graph representation generated in the *Frontend*, and the generated tiling schedule and memory allocation map generated in the *Midend* to create executable code through a series of code generation passes. All deployment targets share the same execution flow, and Deeploy uses a configurable platform abstraction, the *Deployment Platform*, which allows it to steer operators' mapping, optimization, and lowering according to the platform's configuration. An overview of the Deeploy execution flow is shown in Figure 5.1.

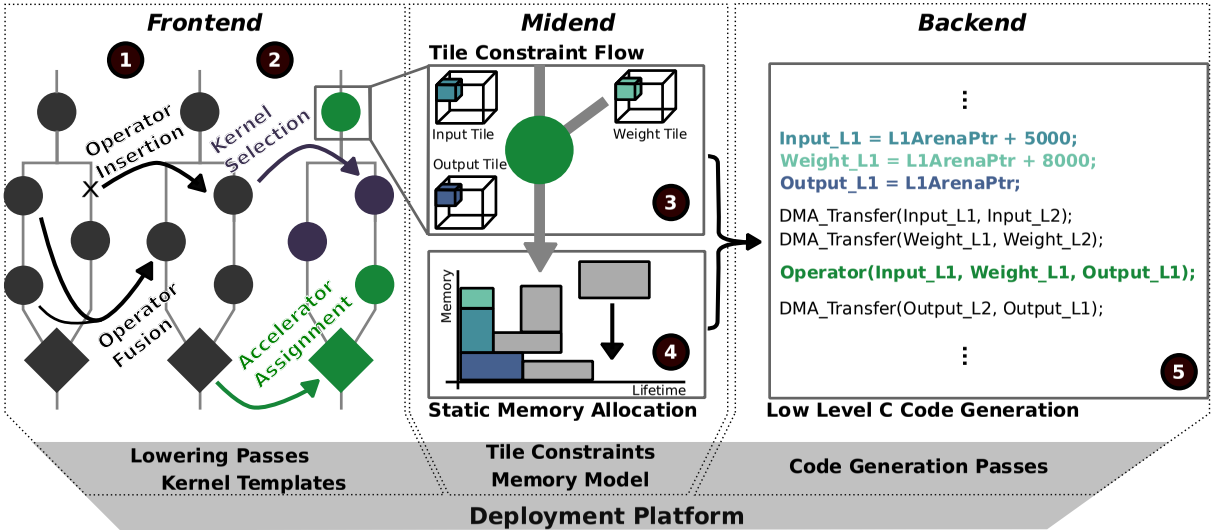

Figure 5.1: Overview of the Deeploy Execution Flow. Steps ① and steps ② are part of the *Frontend*. In the first step, the graph is modified by fusing and inserting platform-specific operators, for example, transposition operators, to match data layout requirements. In the second step, datatypes for every tensor are inferred, the accelerator target is chosen, and kernel templates are selected. The first step in the *Midend*, step ③, is the Tile Constraint Flow, which computes geometrical constraints for the tile sizes of each tensor, adding them to a CP. The resulting tensor size variables are translated into a 2D bin packing problem in step ④. The solution of the co-constrained tiling and static memory allocation problem is computed by the ORTools CP-SAT solver and finally processed in step ⑤ in the *Backend*. Step ⑤ generates platform-specific C Code exploiting DMA transfers. Each step of the execution flow is highly configurable through the *Deployment Platform* object.

5.3.1 Data Structures

Deeploy distinguishes between three types of buffers: *Variable Buffer*s, *Transient Buffer*s, and *Constant Buffer*s. *Variable Buffer*s represent tensors that contain data that is not constant at compile-time, i.e., network inputs, outputs, and intermediate activations. *Constant Buffer*s represent compile-time constant data used in inference, i.e., network weights and other network parameters. Lastly, *Transient Buffer*s represent scratchpad memory locations for kernel execution, e.g., *im2col* buffers for convolution kernels [20, 104], or reorder buffers for efficient transposition kernels. Typically, the amount of space used in *Transient Buffer*s depends on the operator's parametrization, distinguishing them from *Variable Buffer*s. In contrast to simpler DNN topologies, EFMs employ data structures that require advanced allocation strategies, such as the KV caches of autoregressive SLMs, as they have more complex buffer lifetime requirements than intermediate tensors found in CNNs. Addressing these constraints requires a more sophisticated management of the buffers' lifetime and memory allocation than in other deployment tools targeting extreme edge devices [3, 31].

The distinction between global and local section buffers is relevant for code generation; global objects are allocated as global C variables, while local objects are only accessible in the inference code. As such, global variables are alive throughout an inference execution, while local variables are allocated and deallocated as the network's execution schedule requires.

5.3.2 Frontend

Deeploy's *Frontend* is designed around ingesting quantized Open Neural Network Exchange (ONNX) graphs produced by DNN and Transformer quantization tools like Quantlib [227]. Deeploy implements a configurable lowering pass system based on pattern matching of ONNX graphs to enable efficient and customizable graph-lowering strategies. Each lowering pass consists of a user-defined replacement function and a *source pattern*, which describes the sub-graph that should be replaced. Using the replacement function, each lowering pass uses the matched sub-graph to generate a *target pattern*, which

replaces the *source pattern*. Using this system, the first processing step in the *Frontend* is transforming the input graph into a custom, platform-specific ONNX dialect using lowering passes provided by the *Deployment Platform*. The user further defines operator mappings between custom operators and the engines available in the target platform to control the code generation on the level of individual operators.

Common TinyML kernel libraries like CMSIS-NN and PULP-NN [20, 104] offer kernels for fused linear operators and activations, which can be lowered into by matching pairs of linear operators and quantization operators. Besides operator fusion optimization passes, Deeploy also supports the minimization and insertion of data marshaling operators like transpositions to match the data layout requirements of kernel libraries. An example of such an operator insertion pass is adding transpositions operators to optimize the data layout of the B matrix for General Matrix Multiplication (GEMM) kernels of type $Y = \alpha AB + \beta C$ for better data access locality.

The second step after transforming the input graph into the platform-specific dialect in the *Frontend* is parsing, during which every operator in the network is analyzed to construct an initial context of buffers used in the network's execution, and *Type Inference & Kernel Selection* where every buffer in the context is assigned a type. The types used in Deeploy correspond to standard C types (e.g., *int8_t*, *float32*) or custom data types, depending on the kernels used by the *Deployment Platform*. To guarantee a valid type assignment, Deeploy propagates type information top-to-bottom. The user must only provide the input types for every graph's input tensor to achieve this; then, using this information, Deeploy matches the input types of each operator with one of the kernel signatures provided by the *Deployment Platform*.

The final result of the *Frontend* is an assignment of low-level kernel templates to every operator in the lowered platform-specific ONNX, which satisfies the type constraints imposed by the network's operators.

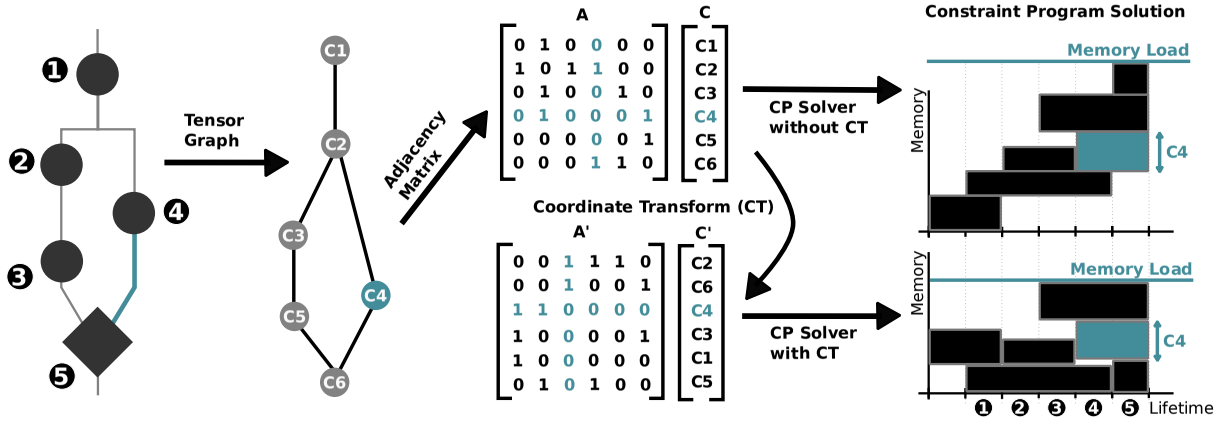

Figure 5.2: Example of the co-optimization of tiling and static memory allocation algorithm for one memory level in Deeploy. First, the lifetime of each tensor in the graph is calculated under the execution schedule shown on the left. Next, the memory scheduler constructs an adjacency matrix of the tensor graph and extracts the cost vector from the tile constraint flow shown in the middle. Finally, Deeploy applies a coordinate transform within the CP. On the right-hand side, the 2D bin packing solution is presented with the naive solution on top, and the solution found by Deeploy is shown below.

5.3.3 Midend

The second stage of Deeploy's execution flow, the *Midend*, receives the platform-specific ONNX graph and the kernel assignment for each operator from the *Frontend*. The *Midend*'s purpose is to perform all optimization operations required to generate low-level optimized C code for the target platform in the *Backend*. The *Midend* is divided into two optimization steps: *Memory Level Annotation* and *Tiling & Memory Scheduling*. To model the CP used to compute the tiling and static memory allocation solution, Deeploy uses Google's ORTools.

Memory Level Annotation

The memory level annotation step annotates every buffer in the compilation context with a memory hierarchy level. The motivation for defining the storage location of every tensor is to model code generation constraints closely to the hardware; most embedded systems designed for TinyML applications use multiple memory or cache levels [25,30] to optimize the trade-off between storage density and memory access latency. While Deeploy supports the tiling of buffers, directly assigning buffers' memory levels to lower cache levels can lead to performance improvements. When targetting accelerators that would otherwise be limited by the available bandwidth towards higher-level caches, controlling memory allocation has a significant performance impact [30].

Tiling

The second processing step in the *Midend* is *Tiling & Memory Scheduling*. For every kernel template chosen in the *Frontend*, the target platform must specify a Tile Constraint (TC). The TC models the geometric and platform-specific constraints for tiling an operator. For a tiling solution to be correct, all geometric constraints must hold. For example, the spatial dimensions of a softmax activation's output tile must be the same as its input tile's dimensions. As such, geometric constraints do not depend on the implementation of an operator. While it is possible to tile large tensor operators down to single instructions when targeting processor cores, the same does not hold for accelerators. Specifying TCs and platform-specific

constraints on a per-kernel basis is especially important for handling the tiling problem for loosely-coupled accelerators since they typically only support specific dimensions to be tiled, owing to their specialized datapaths [25, 30].

Similarly to the *Type Inference & Kernel Selection* flow, the Tile Constraint Flow (TCF) is applied top-to-bottom through the execution schedule of the network, adding the geometric and platform-specific tile constraints of every operator to the CP. Furthermore, the TCF adds one symbolic variable per dimension per tensor in the network to the CP and a symbolic variable for every tensor, representing its size as the product of all dimension variables. Using this formulation, the solution of the CP represents the size of the largest tile.

Memory Scheduling

After the geometrical constraints of every mapped kernel template in the network are collected and added to the CP, Deeploy's memory scheduler calculates the lifetime of every tensor in the network over the user-provided execution schedule of the ONNX graph as shown in Figure 5.2. As previously mentioned, this is an essential step for autoregressive Transformers that must accommodate short-lived tensors (e.g., intermediate activations, residuals) and long-lived buffers (such as KV caches).

Deeploy's memory scheduler computes a tiling path using the *Deployment Platform*'s memory hierarchy model to assign a sequence of memory transfers through the different memory levels. Using the calculated lifetimes and the tensor's size variable computed before, the memory scheduler models the problem of computing a static memory allocation schedule as a 2D bin packing problem [225, 228], where the horizontal axis represents lifetime, and the vertical axis represents memory address space.

Similar to other state-of-the-art algorithms [225], Deeploy's scheduling CP works with Tetris scheduling introduced in TetriSched [229], where memory buffers are scheduled one after another, adding to the maximum load of each of their lifetime's bins. To solve the tiling and allocation problem in a single shot, the memory allocation of each

buffer is coupled to the tiling solution, which requires expressing the order in which they are scheduled within the CP as well.

The first step to modeling the memory allocation problem is to pick a random schedule of memory buffers and compute the adjacency matrix A of the tensor graph. We collect the memory size of each buffer, represented as an integer variable of the CP, in a cost vector C. For any permutation matrix P, $A' = P \times A \times P^T$ is a valid adjacency matrix with associated cost vector $C' = P \times C$. A valid $N \times N$ permutation matrix can be expressed as:

$$
\begin{aligned}
p_{i,j} &\in [0,1] && \forall i,j \in [0, N-1] \\
\sum_{i=0}^{N-1} p_{i,j} &= 1 && \forall j \in [0, N-1] \\
\sum_{i=0}^{N-1} p_{j,i} &= 1 && \forall j \in [0, N-1]
\end{aligned}
$$

Next, the total memory load is computed iteratively using $A' \,\&\, C'$: since we use Tetris scheduling, we add each buffer's memory size to the size of the last scheduled buffer whose lifetime overlaps. We use a vector of intermediate variables containing one entry for each buffer, H, representing the memory load in the lifetime region of each buffer. The vector H is computed as follows:

$$
\begin{aligned}
H_0 &= 0 \\
H_j &= \max_{i=0 \ldots j-1} \left(A'[j,i] \cdot H_i \right) + C'_j
\end{aligned}
$$

The total worst-case memory load for all execution steps is then computed as memory load $= \max_{i=0 \ldots N} (H_i)$.

In contrast to other static memory schedule algorithms, which focus on calculating an optimal solution for memory blocks of fixed size, our algorithm combines the constraints on tile sizes and memory layout calculation into a single CP; this allows Deeploy to simultaneously optimize static memory allocation as well as tile sizing to control memory use during the entire inference process, which is critical to matching the memory constraints of extreme-edge SoCs with the complex buffer lifetime requirements of Transformers. An overview of the

co-constrained tiling and static memory allocation algorithm is shown in Figure 5.2.

5.3.4 Backend

Every kernel template picked in the *Frontend* is assigned a list of code generation passes by the *Deployment Platform*. Each code generation pass operates on a code segment, starting from the original kernel template, and may add to or modify its code segment. Besides enabling integration of custom passes, Deeploy offers standard code generation passes required for generating correct code, e.g., memory allocation and deallocation generation, which inserts calls to heap-based allocators or sets pointers to predefined memory locations calculated during *Tiling & Memory Scheduling*.

An essential set of code generation passes is centered around generating closures for code segments. In the context of Deeploy, closure generation consists of three parts: the closure function itself, which encapsulates a code segment; the closure environment, which contains every free variable used within the code segment and must be passed to the closure function; and the closure invocation, which is either an offloading function or a call to the closure function.

Deeploy implements closures as standard C functions by generating a function call around the target code segment and passing the closure environment as a struct pointer. Deeploy captures the relevant free variable expressions by analyzing the Abstract Syntax Tree (AST) of the underlying code segment using the Mako templating library [230]; since the function signature of the kernel template is known to Deeploy, it can extract arguments used in the kernel template that refer to local buffers, and pass them to the closure using an argument struct. During code generation, the closure generation pass hoists the closure function definition into the global context, inserts code for constructing the argument struct and returns the function call to the hoisted closure as the new code segment for subsequent code generation passes.

An important application for Deeploy's closures is to facilitate operator offloading, which is required for programming processor-based

Kernel Signature

```
// Function signature
void gemv_s8_s8(int8_t* input, int8_t* weight,
                int32_t* bias, int8_t* output,
                uint16_t M, uint16_t N, uint16_t O);
```

Kernel Template

```
// Kernel Template
gemv_s8_s8(${A}, ${B}, NULL, ${C}, 1, ${N}, ${O});
```

Cluster Offloading

Global Definitions

```
typedef struct {
    int8_t* A;
    int8_t* B;
    int8_t* C;
} GEMV_closure_args_t;
void GEMV_closure(void* GEMV_closure_args){
    GEMV_closure_args_t* args =
    (GEMV_closure_args_t*) GEMV_closure_args;
    int8_t* _A = args->A; int8_t* _B = args->B;
    int8_t* _C = args->C;

    gemv_s8_s8(_A, _B, NULL, _C, 1, ${N}, ${O}); }
```

Kernel Replacement

```
// GEMV Closure Call
GEMV_closure_args_t GEMV_closure_args = {
    .A = ${A}; .B = ${B}; .C = ${C};
};
// Parallelize Closure over eight cores
pi_cl_team_fork(GEMV_closure, &GEMV_closure_args, 8);
```

Figure 5.3: Bottom-up offloading closure generation for a GEMV kernel. All arguments that refer to non-global *Variable Buffer*s or *Constant Buffer*s are captured and used to generate a closure struct typedef and a closure function that unpacks the argument struct and calls the original kernel. Finally, the kernel template is replaced with a function *pi_cl_team_fork*, which takes the newly generated closure as an argument and offloads its execution to all eight cluster cores.

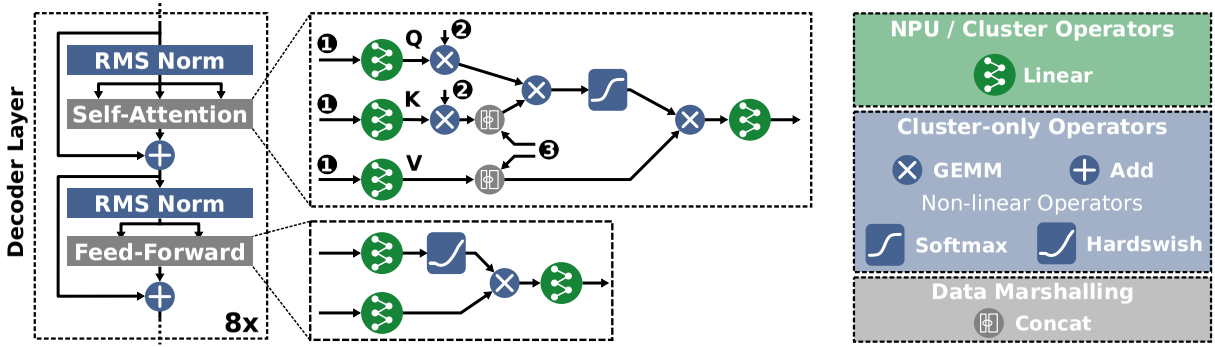

Figure 5.4: Overview of the Llama model deployed in this chapter. The eight decoder layers of the model are shown on the left and consist of an *RMSNorm* - *Self-Attention* - *RMSNorm* - *Feed-Forward* layer stack. Input ① in the self-attention inset corresponds to the token input. Input ② corresponds to the rotational embedding used in Llama models. Input ③ are the KV cache inputs used during autoregressive inference. Notably, during autoregressive inference, the new row of the K and V matrices computed on the input token are appended to the KV cache.

accelerators like compute clusters or loosely-coupled, memory-mapped accelerators like NPUs. An example of closure generation for operator offloading to the octa-core cluster is shown in Figure 5.3.

Tiling code generation is implemented as a pass as well. Deeploy supports DMA engines and uses them in tiling code generation to move tiles between different memory hierarchy levels according to the tiling solution computed in *Tiling & Memory Scheduling*. To hide the latency of DMA transfers, Deeploy can configure tiling for operators to use double-buffering, which constrains the tiling solution to reserve twice the required space for every input- and output tile. During code generation, Deeploy schedules data fetching and writeback to occur in parallel with kernel execution to minimize latency.

5.4 TinyStories Llama Model

As a concrete example of our deployment flow for next-generation EFMs, we quantize and deploy an SLM on a heterogeneous MCU, Siracusa, introduced in Section 5.5. We chose a Llama2 model pre-trained on the tinyStories dataset [209] from HuggingFace[2], with a hidden size $d_m = 64$, $h = 16$ parallel attention heads, $N = 8$ layers and an intermediate size $d_{ff} = 256$ for the feed-forward layer. The model architecture is shown in Figure 5.4. Note that, however, any SLM fitting the memory constraints of the target platform can be deployed with the same flow.

Like all other decoder-based language models, the Llama model we use in this Chapter has two fundamental inference modes, which we refer to as autoregressive inference mode and parallel inference mode, and generates its response in two distinct phases, the *prompting phase* and *generation phase*; the prompting phase ingests the initial sequence of user input tokens, whereas the generation phase generates the model's output tokens autoregressively.

[2]https://huggingface.co/Maykeye/TinyLLama-v0

5.4.1 Prompting Phase

Inferences follow a two-pass regime: First, the text input is translated into a sequence of tokens, typically referred to as the prompt. The prompt can have an arbitrary sequence length S_p, up to the size of the context window of the model.

In the first pass of the model, the prompt is processed to produce the first output token. Since all tokens of the prompt are available ab initio, the decoder can process them in a parallel single-shot fashion by applying causal-masking of the attention matrix [108]. This first pass generates the first token output and the K and V matrices, which may be reused in the subsequent *generation phase*. This process parallels the function of encoder layers used in the first Transformer models [108].

5.4.2 Generation Phase

In the generation phase of the inference process, output tokens are generated one at a time using the previous token outputs as the model's input. While every step of the generation phase may use the same parallel inference mode described in the previous Section, doing so would require recomputing all previous tokens' K and V submatrices. Therefore, the K and V matrices of previous inference steps are typically cached in memory to avoid the quadratic cost of recomputing them [108].

As the parallel inference mode and autoregressive inference mode require different trade-offs in memory allocation for KV caching and storage of intermediate results we deploy them using separate ONNX models which reflect these trade-offs: For the parallel inference mode we export an ONNX model with a single input and output for the token sequence and outputs for the computed KV submatrices which are stored for the next generation phase. For the autoregressive inference mode, we use an ONNX model that additionally requires cached KV submatrices. While computing outputs using KV caches is significantly more efficient regarding the absolute number of operations, loading and storing the KV caches induces significant data movement,

and the smaller operator dimensions make the generation phase much more challenging to accelerate.

5.4.3 Quantization Setup

To quantize the SLM for deployment on extreme edge devices with integer-focused SIMD processors and DNN accelerators, we used QuantLib [227] with the Trained Quantization Thresholds (TQT) algorithm for PTQ [18]. QuantLib inserts requantization layers after operators which results in higher bitwidth outputs. Furthermore, it harmonizes scaling factors for operators like addition and concatenation and replaces various operators with their quantization-aware equivalents. Following this, we use a single token to execute PTQ over three inference epochs. Initially, we collect statistics to initialize the clipping bound for all activations and weights. At the end of the second epoch, we quantize all linear operations, and in the final epoch, we quantize non-linear operations, including Softmax and RMSNorm. Subsequently, the model is projected to the integer domain and exported as an ONNX graph. To leverage the advanced hardware support for SIMD operations in the PULP Cluster and Siracusa's NPU, *N-Eureka*, we chose to quantize all activations and weights used in matrix multiplication to 8 bit integer precision. We use I-BERT's approximation for Softmax [128] and the Hardswish approximation for Swish activations [221]. Moreover, we perform all divisions in RMSNorm [231] layers with 32 bit numerators and denominators to preserve accuracy.

5.5 Deployment Platform

This Section introduces the hardware platform used in this Chapter as a deployment target to deploy the SLM introduced in Section 5.4 and goes over the NPU-specific *Deployment Platform* implementation in Deeploy.

5.5.1 Siracusa

Siracusa [30], is a low-power, heterogeneous RISC-V MCU implemented in TSMC 16 nm technology, which is the multi-accelerator SoC targetted in this Chapter. Siracusa is designed for efficient AI inference, which can leverage its dedicated NPU, *N-Eureka*, and generalistic DSP tasks, which can exploit both dedicated XpulpNN ISA extensions [20] enabling SIMD processing of low-precision integers, as well as an accelerator cluster of eight RISC-V cores which enable Single Program Multiple Data (SPMD) processing.

To enable single-latency access from cluster cores to the L1 Tightly-coupled Data Memory (TCDM), all cores and the 16 L1 memory banks are connected through a TCDM interconnect using one 32-bit port each, granting a total memory bandwidth of 256 bit/cycle to the compute cluster. The cluster's TCDM memory banks are also accessible from the *N-Eureka* accelerator using 9-bank wide, 288 bit accesses. To manage contention on accesses to the single-ported memory banks, Siracusa integrates a lightweight, programmable access arbiter, which allows the set the maximum number of stall cycles for the accelerator; if accesses from the core-side interconnect cause accelerator access to stall for the programmed number of cycles, the arbiter will stall core accesses and grant it to *N-Eureka*.

The *N-Eureka* accelerator uses a mixed-weight-precision bit-serial datapath, which is optimized for executing dense 3×3, depthwise 3×3, and dense 1×1 convolution operations with 8 bit activations and 2 bit to 8 bit convolution weights [30]. To support the bit-serial nature of the datapath, *N-Eureka* requires its weights to be stored in a non-standard bit-interleaved data format, which requires offline transposition, padding, and bit shuffling of CNN weight tensors. *N-Eureka* is designed as an output-stationary accelerator, opting to cache small input tiles and streaming weights. To execute operations larger than its internal buffers, it integrates a hardware tiler with a programmable number of tiles and strides between dimensions and fixed tile sizes that match the buffer sizes. To increase the available memory bandwidth for *N-Eureka*'s weights and minimize off-chip access to fetch weights, the cluster integrates a Neural Memory Subsystem (NMS), which contains two dedicated 4 MiB memory subsystems, implemented in

SRAM and Magnetoresistive Random Access Memory (MRAM) technology respectively, which are designed to hold weights for the *N-Eureka* accelerator and are attached through a dedicated 256 bit/cycle weight data port.

The compute cluster and *N-Eureka* are located in a shared clock domain, the heterogeneous cluster, which communicates with the rest of the SoC, mainly consisting of a controller core, 2 MiB L2 memory, and peripherals, through a 64 bit wide Advanced eXtensible Interface Bus (AXI) bus, which can be used by a DMA integrated within the cluster, to transfer data between the L1 and L2 memories autonomously.

While Siracusa is equipped with significant computing capabilities through two dedicated accelerators and sizeable on-chip memory, deploying an advanced neural network on this device is a challenging problem. While weight storage for layers that can be executed on *N-Eureka* is plentiful, all other layers' activation, weight, and output tensors must be tiled to fit within 256 KiB of L1 memory. Furthermore, memory transfers between L2 and L1 should be orchestrated using the DMA to minimize stalling.

5.5.2 Deeploy Integration

We address the deployment challenges posed by Siracusa's heterogeneity through an augmented *Deployment Platform* model. This subsection gives an overview of the additions implemented to use Deeploy for deploying SLMs on Siracusa and, more generally, of the modifications needed to support a generic new platform in our deployment tool.

As Deeploy's core primitives are optimized kernels, we chose the PULP-NN [20] kernel library, which integrates parallel kernels as well as single-core implementations, as our target for utilizing the octa-core cluster. The PULP-NN kernels focus on efficient implementations of fused linear and quantization layers. We support fused layers through lowering passes that match the supported operator combinations and merge them in the *Frontend* of Deeploy. We further added fused linear operator TCs, which add kernel-specific constraints besides providing general geometric constraints.

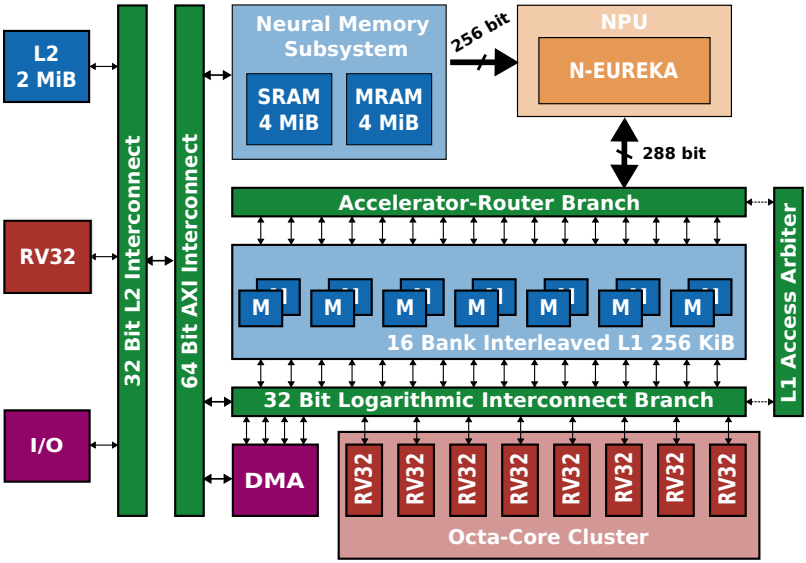

Figure 5.5: Overview of the Siracusa SoC featuring its DSP-enhanced octa-core RISC-V cluster and host controller (red), NPU (orange), complex memory hierarchy with two levels of scratchpad memory and a Neural Memory Subsystem (blue), two arbitrated interconnects towards the L1 memory and an AXI interconnect (green), and peripherals such as the cluster DMA and chip-level I/O (purple).

We implement function offloading to both the NPU and the octa-core compute cluster in Siracusa using the closure system, as detailed in Section 5.3.4.

The *N-Eureka* accelerator provides greater compute capabilities than the octa-core cluster for CNN operators, achieving a peak throughput in the range of hundreds of GOp/s for pointwise and 3×3 convolutions. Even though SLMs do not employ these types of operations, we add a custom *linear layer to pointwise convolution* lowering pass that converts GEMM operators with compile-time constant weight matrices into pointwise convolutions. This method allows us to deploy all linear layers in Transformer models, as shown in Figure 5.4, on the NPU.

For this lowering pass, we consider GEMM operation of type $Y = \alpha AB + \beta C$, where Q are appropriately integer-quantized numbers:

$$A \in Q^{M \times N} \qquad B \in Q^{N \times O} \qquad C \in Q^{M \times O} \qquad Y \in Q^{M \times O}$$

Similarly, we define the pointwise convolution operator as $Y = A \otimes B + C$, with the same dimension definition used in PyTorch:

$$A \in Q^{H \times W \times C_{in}} \qquad B \in Q^{C_{out} \times 1 \times 1 \times C_{in}}$$
$$C \in Q^{H \times W \times C_{out}} \qquad Y \in Q^{H \times W \times C_{out}}$$

We map the dimensions of the pointwise convolution to those of a GEMM operation by setting $H := 1$, $W := M$, $C_{in} := N$, and $C_{out} := O$. For this mapping to succeed, the C operand in the GEMM operation must be reducible to a dimension of $[\,1 \times O\,]$, i.e., all rows in the matrix are identical.

Lastly, we annotate all pointwise convolution weights previously transformed from the GEMM operators to be allocated in the NMS, allowing the accelerator to leverage its significantly larger bandwidth.

5.5.3 Deployment Setup

As explained in Section 5.4, the dual inference modes of decoder-only models require different deployment strategies, as the autoregressive

Table 5.1: Compiler performance metrics for the 128th autoregressive inference step using the *NPU with NMS Deployment* scenario with varying numbers of decoder layers

Number of decoder layers	1	2	3	4	5	6	7	8
Number of operators	32	64	96	128	160	192	224	256
Deeploy compilation time [s]	1	2.65	5.17	7.92	11.72	16.9	22.25	28
Text section size [kB]	42.4	61.8	84.4	105.7	128.1	150.1	171.5	194.8
Data section size [kB] (input and output buffers)	10.4	18.6	26.8	35.0	43.2	51.4	59.5	67.7
Data section size [kB] (requantization parameters)	7.0	15.3	23.6	31.9	40.2	48.5	57.8	66.1
Weight memory section size [kB] (pointwise convolution parameters)	64	128	192	256	320	384	448	512

inference mode requires significant memory for KV caching. We deploy two model prototypes to accommodate this difference, one for autoregressive inference mode and one for parallel inference mode.

The autoregressive inference mode model uses additional network inputs corresponding to the previous sequences' KV caches. Other than that, the deployment setup between both models is equal. We allocate all graph inputs and outputs as global *Variable Buffer*s in Siracusa's L2 memory, and annotate all local *Variable Buffer*s modeling intermediate tensors in L2 as well. In deployment scenarios that use Siracusa's NMS, we allocate all linear layer weights in the NMS but use L2 for all activations.

Unless stated differently, all network operators are executed on the cluster and use Deeploy's TCF to generate tiled inference code, which orchestrates transfers of input, weight, and output tensors between the L2 memory and the L1 memory. For operators executed on the NPU, weights are stored in the NMS in their entirety and ingested by the accelerator without moving them into L1 first, leveraging the increased available bandwidth from the NMS.

5.6 Results

This section discusses the measurement results of deploying the TinyStories SLM on Siracusa and benchmarking results of general Transformer layers. First, we present the quantized model's accuracy results. Next, we discuss the setup used to measure performance results on Siracusa. Finally, we present our benchmarking and end-to-end silicon measurements.

5.6.1 Quantization Results

We consider the average accuracy in a zero-shot setting to evaluate our quantized model on various tasks. We consider the Hellaswag [232], OpenBookQA [233], WinoGrande [234], ARC-Easy and ARC-Challange [235], BoolQ [236], and PIQA [237] datasets and use the Language Model Evaluation Harness [238]. The comparative results, displayed in Table 5.2, clearly indicate that our quantized model

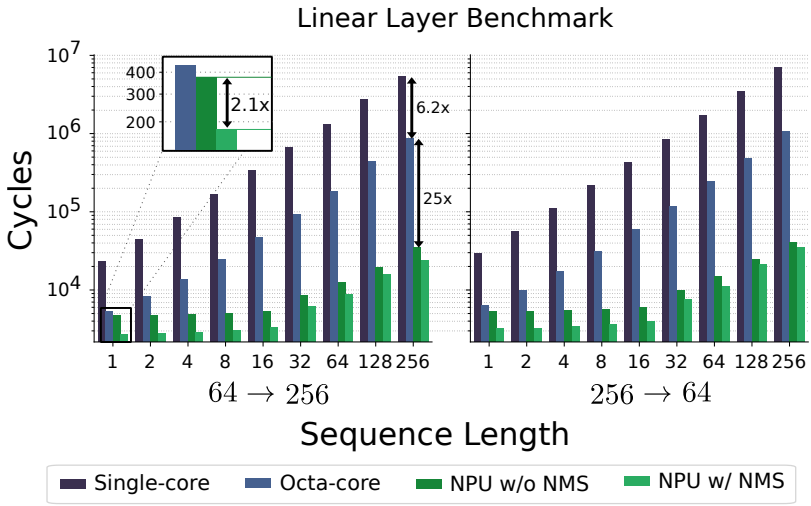

Figure 5.6: Performance results for linear layer operators offloaded on *N-Eureka* using Deeploy code generation. The highlighted inset shows that the NMS' added storage and bandwidth leads to performance gains of up to 2.1 × in memory-bound operator configurations. In large linear layer configurations, the speedup achieved by the NPU is 25 × compared to the octa-core implementation, and another 1.6 × when using the NMS for weights.

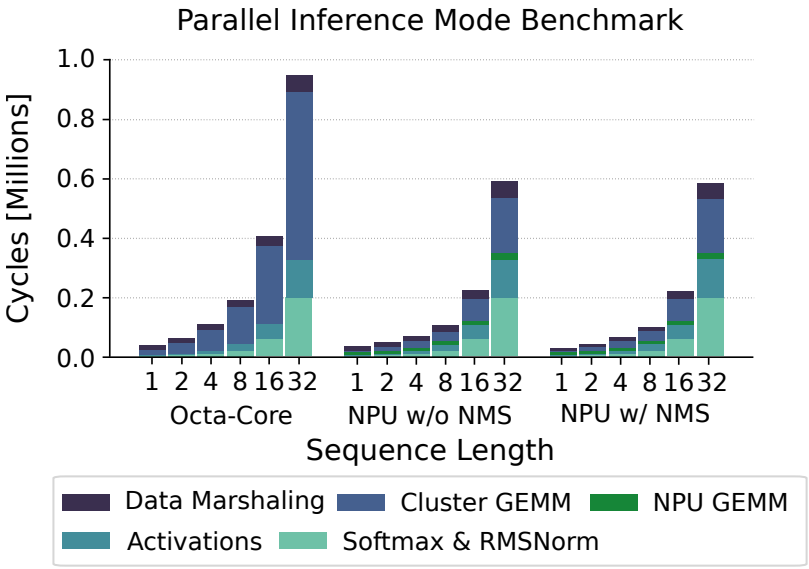

Figure 5.7: Cycle breakdown of parallel inference in the studied SLM. Due to the larger contribution of operations from matrix multiplications using the NMS performance of offloaded GEMM operators increases by 17.8 ×, and end-end-performance improves by 61 % for sequence length 32 while maintaining low overheads of only 9 %, even when fully leveraging both the cluster and NPU.

Table 5.2: Comparison of Average Accuracy

Model	Type	Avg Accuracy (%)
TinyLlama-5M	BF16	31.59 ± 1.09
TinyLlama-5M	**INT8 PTQ**	**32.15 ± 1.11**
TinyStories-GPT2-3M	FP32	32.29 ± 1.11
TinyStories-3M	FP32	32.61 ± 1.10
TinyStories-8M	FP32	36.79 ± 1.13
TinyStories-33M	FP32	35.63 ± 1.13
GPT-Neo 125M	FP32	40.39 ± 1.15
MobileLLM 125M	FP32	47.00
TinyLlama-1.1B-Chat-v0.4	FP32	49.41 ± 1.23

maintains its accuracy, even showing a slight increase compared to the standard full-precision baseline model.

To give an idea of the accuracy levels achieved by other SLMs with similar complexity, we also include accuracy results in Table 5.2 for different TinyStories model [209] and a variant using the GPT-2 architecture[3], GPT-NEO [239], MobileLLM [240], as well as the significantly larger TinyLlama [210].

5.6.2 Deployment Evaluation Setup

To evaluate the model's performance in autoregressive mode and for causally masked parallel inference, we measure each inference step individually with code generated by Deeploy. We start from empty KV caches for causally masked parallel inference and process N input tokens simultaneously. We start from the KV caches of the previous inference step for all experiments in autoregressive mode. While Deeploy supports longer sequences, we follow the convention set by Eldan et al. and limit the context window to 256 tokens [209]. To calculate the average throughput and energy per token, we take the average over all 256 inference steps.

[3]https://huggingface.co/calum/tinystories-gpt2-3M

We report all power numbers measured on a Siracusa prototype board using a Keysight N6715C DC, supplying all operating voltages and measuring current. We perform all experiments under nominal conditions, i.e., 0.8 V supply voltage and 360 MHz operating frequency of the cluster domain. We measure power consumption for every inference by averaging the power consumption of the model run in a continuous loop.

We measure four distinct deployment scenarios: In the first scenario, *Single Core Deployment*, we only generate code using a single RISC-V core. In the second scenario, *Octa-Core Deployment*, we generate code using all eight RISC-V cores of the cluster without using the NPU. In the third scenario, *NPU without NMS Deployment*, we generate code using all eight RISC-V cores and *N-Eureka* without offloading weights to the NMS. In the last scenario, *NPU with NMS Deployment*, we generate code using all eight RISC-V cores and *N-Eureka* with the NMS. We use the Siracusa *Deployment Platform* in Deeploy to generate code for all scenarios.

5.6.3 Microbenchmarking Results

To validate our approach of offloading GEMM operators on *N-Eureka*, we first measure the performance of *N-Eureka* and the RISC-V cluster on GEMM kernels. Specifically, we study the performance of the Q, K, and V projections in the attention layer and linear layer performance in feed-forward layers for different sequence lengths S in parallel inference mode. For the Llama model we study in this chapter, these projections use dimensions $256 \rightarrow 64$ and $64 \rightarrow 256$. Our measurements are shown in Figure 5.6.

Transitioning from single-core to octa-core cluster execution, we measure a performance improvement of 6.2 ×, thanks to the low-overhead parallelization on the cluster cores. Transforming the linear layer operators into pointwise convolutions, as explained in Section 5.5.2, enables execution on the NPU, which reduces latency by 25 × compared to the octa-core implementation due to the NPU's significant compute resources for convolution operations. Furthermore, we reduce data movement by allocating the convolution weights to the NPU's NMS, increasing the effective memory bandwidth available to

N-Eureka. These optimizations improve performance, especially on memory-bound tasks, like linear layers in attention blocks with low sequence length, by 2.1 × compared to NPU execution without the NMS.

We further profile the execution performance of a representative encoder layer as commonly found in non-regressive Transformer models. For our benchmarking, we chose a configuration with hidden size $d_m = 64$ and $h = 16$ parallel attention heads and an intermediate size $d_{ff} = 256$, paralleling the decoder layer in Figure 5.4. We measure an increase in throughput of 17.8 × when leveraging the NPU to compute linear layers, improving end-to-end performance for encoder layers by 61 %. We further quantify the overheads due to tiling and data marshaling overheads, measuring an end-to-end overhead of only 9 %.

5.6.4 Compiler Evaluation

To study the scalability of our code generation approach, we break down the SLM network, measuring compiler metrics for varying network depth. For each configuration, we profile code size, constant data size, and input- and output buffer size for a single inference step in *NPU with NMS Deployment*, namely the 128th autoregressive inference step. We generate all code with Deeploy and compile the resulting C Code using clang-15. Our results are shown in Table 5.1.

We notice that while code size grows proportionally to the number of operators in the workload, the total size of the binary is dominated by weight storage in the NMS. We further see that while compilation time grows superlinearly with the number of operators in the network, the maximum compilation time of 28 s does not pose a bottleneck for practical purposes.

5.6.5 End-to-end Deployment Results

We thoroughly evaluate the SLM deployed on Siracusa by benchmarking the two operating phases required to execute SLM, namely the *prompting phase* and the *generation phase*.

Table 5.3: Cumulative latency and energy for a 256-step inference of the SLM on Siracusa using the NPU with NMS

	Parallel Inference	Autoregressive Inference	Speedup & Energy Reduction Ratio
Latency [s]	17.6	0.75	23 × faster
Energy [mJ]	3193	125	26 × more efficient

Table 5.3 displays the cumulative runtime and energy for executing a 256-step inference in parallel mode and in autoregressive mode, where KV caching is used. The autoregressive mode outperforms the parallel mode, achieving a 23 × speedup and a 26 × improvement in energy efficiency. These improvements directly result from avoiding the costly recomputation of KV matrices. Averaging the autoregressive inference mode's cumulative latency and energy over 256 steps, we achieve an average throughput of 340 token/s at an average energy cost of 490 µJ/token.

Since the autoregressive mode maximizes the data reuse across the whole inference process, this mode can be considered both during the *prompting* and *generation* phases detailed in Section 5.4. However, this strategy leads to sub-optimal results as running in parallel mode for the *prompting* phase enables better utilization of the NPU without excessive recomputation of KV matrices, as tokens are not fed back in this phase.

The parallel inference mode's performance for the SLMs studied in this work follows the trend of the benchmark shown in Figure 5.7. While we benchmark the end-to-end performance of decoder-only models in this work, the results in Figure 5.7 also apply to encoder-based transformer models, as the parallel inference mode is equivalent to encoder layer execution in such networks. In autoregressive mode the speedup achieved by employing the NPU is only 19 %, which can be attributed to the mode's smaller operator sizing, leading to stalling of the accelerator due to reconfiguration overheads. Additionally, the average proportion of time spent for data marshaling is 40 % for the autoregressive versus just 14 %

for the parallel modes, underlining the memory access intensity inherent to KV caching, which drastically reduces the number of computations leading to reduced arithmetic intensity. A detailed analysis of runtime and breakdown of operator intensity for end-to-end autoregressive inference is shown in Figure 5.8 plots ① and ②.

5.6.6 Deployment Overheads

An important metric for the quality of generated code is the utilization of the system's compute engines. To profile the quality of our code, we measured the overheads incurred by Deeploy for each autoregressive inference step in *NPU without NMS Deployment* and *NPU with NMS Deployment*, shown in Figure 5.8, plot ③. The main difference between the two scenarios is whether Siracusa's NMS is used for compile-time constant GEMM weights. While the reduction in overheads decreases from 33 % to 7 % with increasing sequence lengths and arithmetic intensity, the weight memory drastically reduces the relative time spent on data movement in the first steps of inference. This reduction of overheads is a crucial advantage of the bottom-up compilation approach employed by Deeploy; while other compilers might not consider low-level architectural features like memory hierarchy or only expose a simplified model, Deeploy allows complete control over memory allocation and code generation to leverage knowledge of the target architecture fully.

5.6.7 Comparison with tinyML Compilers

While we designed Deeploy to deploy state-of-the-art and emerging SLMs, we also report results on more classical CNN and ANN workloads, as defined in the MLPerf tiny benchmark [241]. We compare Deeploy with the state-of-the-art open-source Dory tool [3] using the same open-source CNN kernels for PULP MCUs [20] we used in this work. To ensure a fair, compiler-focused comparison, we do not use the NMS or the NPUs of Siracusa. In this mode, both compilers only deploy cluster kernels with equivalent memory constraints. As a third data point, we add measurements of Deeploy-generated code on Siracusa when using the NMS and NPU. Our results are shown in

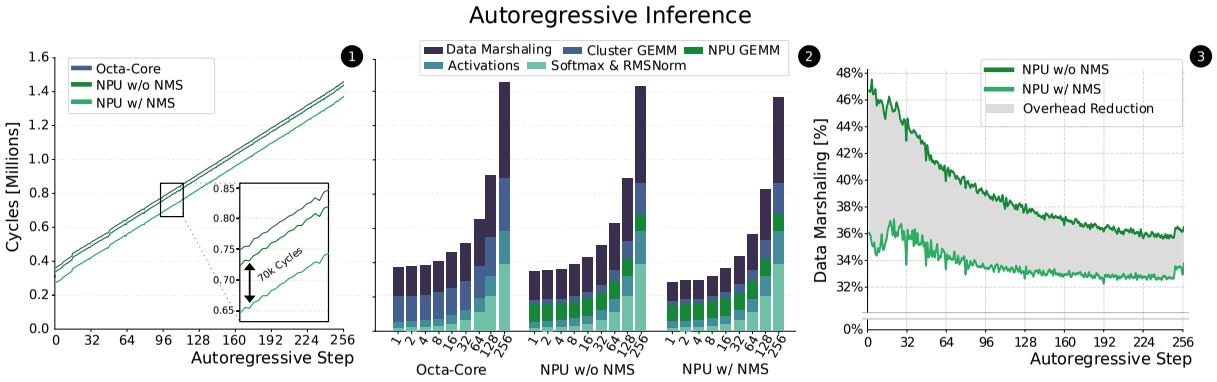

Figure 5.8: Performance results of end-to-end autoregressive inference. Plot ① shows the runtime of each autoregressive inference step in three scenarios corresponding to *Octa-Core Deployment*, *NPU without NMS Deployment*, and *NPU with NMS Deployment*. The plot shows that autoregressive inference on Siracusa is highly memory-bound in all scenarios, which is due to the transfer of KV caches between L2 and L1; *NPU with NMS Deployment* reduces the runtime of every step by approximately 70 kcycles since weights are stored untiled in the NMS, reducing the required L2 - L1 data transfers. The second plot ② shows a breakdown of the runtime in the different operators of the network and data marshaling overheads. Evidently, the higher compute throughput of *N-Eureka* is unused due to the overall memory-boundedness. Finally, plot ③ shows that the data movement overhead reduction afforded by the NMS decreases with increasing sequence lengths as the overhead of transferring KV caches increases.

Table 5.4: Latency results of Dory and Deeploy on the MLPerf Tiny benchmark, running on Siracusa at a clock frequency of 360 MHz.

Benchmark	Siracusa w/o NPU Dory	Siracusa w/o NPU Deeploy	Siracusa w/ NPU Deeploy
DS-CNN	1.4 ms	1.4 ms	0.39 ms
MobileNetv1	5.6 ms	5.6 ms	0.69 ms
ResNet	3.7 ms	3.7 ms	0.60 ms
ToyAdmos	0.24 ms	0.24 ms	0.11 ms

Table 5.4. We find that Deeploy generates code with an equivalent latency of Dory up to 1 % of variation, underlining that even though Deeploy chooses a more general compilation approach than Dory, it does not incur any performance penalties.

5.6.8 Comparison with the State-of-the-art

Currently, most efforts on EFM deployment target models with more than a billion parameters on high-end Microprocessors (MPUs) and embedded processors such as the I.MX95 or NVIDIA Orin or mobile phone chips, featuring multi-GiB external memories and multi-W power envelopes [242, 243]. Even though our performance and efficiency are extremely competitive, quantitative comparisons against these deployments would be unfair in our favor as we target much smaller SLMs.

Considering SLMs in the 100s million parameters range, we compare our implementation on Siracusa with another small-scale Llama model for edge devices, MobileLLM, by Liu et al. [240]. Liu et al. deploy a 125 MParameter SLM on an iPhone 13 featuring an A15 Bionic chip in 5 nm technology using the highly optimized Metal Performance Shaders (MPS) backend for Apple devices, achieving a throughput of 64 Token/s. While their paper does not profile the exact energy consumption of their models during inference, Liu et al. optimistically estimate the energy consumption of their setup with 12.5 mJ per token. Compared to this estimate, our implementation uses 26 × less energy per token while achieving 5 × more throughput, for a total 130 × higher energy efficiency. When normalizing throughput with the number of operations per token of their network, we find that they achieve an equivalent of 4800 TinyStories Llama tokens per second. Under this estimate, our end-to-end energy efficiency on Siracusa implemented in an older 16 nm TSMC technology node is 1.7 × higher.

A comparison with a similar-scale (10s million parameters) model as ours is possible against the *llama2.c* [244] implementation of the TinyStories-15M model on a Samsung Galaxy Watch 4, demonstrated to achieve 22.1 Token/s [245] using an Exynos W920 dual-core ARM Cortex-A55 processor [246]. Neglecting the power consumption of Dynamic Random Access Memory (DRAM) accesses, only considering a

power consumption of 300 mW per core in Samsung 5 nm technology [247], we estimate the power consumption during inference as 600 mW. Under this assumption, the Galaxy Watch 4 achieves an energy efficiency of 27 mJ per token, 55 × lower than ours. Normalizing for operations per token, our energy efficiency is 13.4 × greater, even though the Exynos W920 is implemented in an advanced Samsung 5 nm technology node.

5.7 Conclusion

In this Chapter, we presented Deeploy, a novel compiler for DNNs allowing broad customizability of deployment flows. We presented the integration of Siracusa, a heterogeneous RISC-V SoC featuring an octa-core compute cluster and an NPU. We demonstrate the deployment of a SLM trained on the TinyStories dataset on Siracusa, achieving a state-of-the-art throughput of 340 Token/s at an average energy cost of 490 µJ per token in autoregressive inference mode by efficiently leveraging on-chip KV caching.

We further analyzed in detail the efficiency of our generated code via microbenchmarks, achieving data marshaling overheads of only 9 % on Transformer encoder layers, even when fully utilizing both cluster cores and NPU collaboratively.

Lastly, we demonstrated that while data marshaling overheads are significant in the autoregressive inference mode, the energy savings compared to executing the generation phase of SLM in parallel mode outweigh this drawback, reducing the energy cost per token by 26 × while increasing throughput by 23 ×.

In future work, we plan to leverage Deeploy's flexibility to support emerging computer architecture innovations, such as multi-accelerator SoCs integrating Compute-In Memory (CIM) macros. Furthermore, we'll investigate approaches to address the data movement challenges posed by the autoregressive inference mode more effectively.

Chapter 6

Conclusion

This chapter summarizes the contributions presented in the previous chapters, concluding this thesis.

6.1 Main Results

Radar-based Gesture Recognition for Extreme Edge Devices

We presented a dual-stage algorithm for classifying gestures from radar data consisting of a *per-frame* CNN and a *per-sequence* TCN. Using this dual approach of extracting spatial features from a CNN and temporal features from a TCN not only minimizes the model's memory footprint, compressing it to 92 kB but outperforms larger, traditional autoregressive models like LSTMs. We demonstrated end-to-end deployment and integration with the sensor on the GAP8 MCU, achieving real-time prediction in a power envelope of only 21 mW.

Linear Transformers for Embedded Keyword Spotting

We introduced a novel neural network architecture based on linear Attention, the WaveFormer, which enables long-sequence Transformer inference on TinyML-class devices. We demonstrated that the architecture outperforms the state-of-the-art networks based on spectrogram analysis by ingesting raw waveforms, achieving 98.8 % and 99.1 % accuracy on the Google Speech V2 12 and 35 class datasets while requiring 2.5,× fewer operations and 4.7 × less parameters. We further described a quantization algorithm for this network and deployed the network on an ARM microcontroller, achieving real-time inference performance at a power consumption of 11.7 mW.

Energy-efficient Acceleration of TNNs

We presented CUTIE, a fully ternary CNN accelerator focusing on minimizing non-computational energy and switching activity so that dynamic power spent on storing intermediate results is minimized. We demonstrate the end-to-end energy efficiency impact of CUTIE's unrolled datapath architecture, the reduced switching activity in adder trees due to sparsity in TNNs, and optimized sparsity-aware training methods. Overall, CUTIE and its TCN extensions achieve a peak energy efficiency of 1036 TOp/J, outperforming the state-of-the-art silicon-proven TinyML quantized accelerators by 1.67 × while achieving competitive accuracy.

Deployment Algorithms for Heterogeneous SoCs

We addressed the challenges of DNN deployment on heterogeneous tinyML SoCs with Deeploy, a bottom-up compiler, allowing user-provided kernel libraries to be integrated. We demonstrate end-to-end deployment of an autoregressive SLM for the first time on a MCU using on-chip memory only by using Deeploy's novel single shot tiling

and static memory allocation algorithm, achieving leading-edge energy efficiency of 490 µJ/Token, at 340 Token/s.

6.2 Future Work and Outlook

In this Section, we give an overview of possible continuations of the work done in this thesis based on the results presented. At the end of this Section, we provide an outlook and conclude this thesis.

Aggressive Quantization Techniques for Attention

While CUTIE provides state-of-the-art energy efficiency and the training and quantization methods presented in this thesis allow for training ternary CNNs, they are not immediately applicable to the Attention mechanism in Transformers. Exploring aggressive quantization for Attention could enable significant energy savings in Transformers, which are paramount to drive adoption of this class of networks in TinyML devices.

Foundation Models for the Extreme Edge

The prevailing trend in the ML community is pointing towards FMs, but the TinyML research community has not yet adapted them. The reasons for this technology gap are manifold, but an important reason is that current-generation FMs use billions of parameters and inherit their model structure from NLP. Studying FMs from the perspective of TinyML, considering sensor-driven, application-specific input encoding and aggressively reducing their size could facilitate adoption and, in turn, drive software and hardware specialization to unlock improvements in inference energy efficiency.

Automatic Compiler Tuning for Heterogeneous SoCs

Deeploy enables exploration of DNN deployment strategies on heterogeneous SoCs. Still, it works in a feed-forward operating mode, which limits the practical feasibility of exploring the exponential operator mapping design space. Especially when scaling TinyML SoCs to multiple accelerators, a broader set of optimization techniques and exploration methods are needed to generate optimal code.

Outlook

In the next step, we aim at extending Deeploy with support for CIM accelerators and multi-cluster systems and improve the tools' mapping and scheduling optimization methods. We further believe that by exploring ternary-weighted Transformers and Foundation Models, we can enable accelerator performance on a similar level to CUTIE, enabling a new approach for Foundation Models for the extreme edge.

Acronyms

AI Artificial Intelligence

ANN Artificial Neural Networks

AR Augmented Reality

ASIC Application-specific Integrated Circuit

ASR Automatic Speech Recognition

AST Abstract Syntax Tree

AXI Advanced eXtensible Interface Bus

BNN Binary Neural Network

CIM Compute-In Memory

CLCA Convolutional Linear Cross-Attention

CNN Convolutional Neural Network

CP Constraint Program

DFT Discrete Fourier Transform

DMA Direct Memory Access

DNN Deep Neural Network

DRAM Dynamic Random Access Memory

DSP Digital Signal Processing

DVS Dynamic Vision Sensor

EFM Embodied Foundation Model

FC fabric controller

FLL Frequency-Locked Loop

FM Foundation Model

FMCW Frequency-Modulated Continuous Wave

GEMM General Matrix Multiplication

GPGPU General-Purpose Graphics Processing Unit

GPU Graphics Processing Unit

GSC Google Speech Commands

HAR Human activity recognition

HCI Human-computer Interface

HMM Hidden Markov Models

HPC High-Performance Computing

IMC In-Memory Computing

IoT Internet of Things

ISA Instruction Set Architecture

KWS Keyword Spotting

LLM Large Language Model

LSTM Long Short-Term Memory

MAC Multiply-Accumulate

MCU Microntroller

ML Machine Learning

MMU Memory-Management Unit

MPS Metal Performance Shaders

MPU Microprocessor

MRAM Magnetoresistive Random Access Memory

NLP Natural Language Processing

NMS Neural Memory Subsystem

NN Neural Network

NPU Neural Processing Unit

OCU Output Channel Compute Unit

ONNX Open Neural Network Exchange

OS Operating System

PTQ Post-Training Quantization

QAT Quantization-Aware Training

RADAR Radio Detection and Ranging

RFDM Range Frequency Doppler Map

RNN Recurrent Neural Network

RRF RADAR Repetition Frequency

RRI RADAR Repetition Interval

SCM Standard Cell Memory

SIMD Single Instruction Multiple Data

SLM Small Language Model

SNN Spiking Neural Network

SoC System-on-chip

SOR Signal-over-Range

SOT Signal-over-Time

SPMD Single Program Multiple Data

SRAM Static Random Access Memory

TC Tile Constraint

TCDM Tightly-coupled Data Memory

TCF Tile Constraint Flow

TCN Temporal Convolutional Network

TNN Ternary Neural Network

TQT Trained Quantization Thresholds

VR Virtual Reality

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012.

[2] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 1–14.

[3] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "DORY: Automatic End-to-End Deployment of Real-World DNNs on Low-Cost IoT MCUs," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1253–1268, Aug. 2021, [TLDR] This work proposes DORY (Deployment Oriented to memoRY) – an automatic tool to deploy DNNs on low cost MCUs with typically less than 1MB of on-chip SRAM memory and releases all the developments – the DORY framework, the optimized backend kernels, and the related heuristics – as open-source software.

[4] F. Conti, L. Cavigelli, G. Paulin, I. Susmelj, and L. Benini, "CHIPMUNK : A Systolically Scalable 0.9 mm2, 3.08 Gop/s/mW @ 1.2 mW Accelerator for Near-Sensor Recurrent

Neural Network Inference," *2018 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, Apr. 2018, [TLDR] CHIPMUNK, a small (<1mm2) hardware accelerator for Long-Short Term Memory RNNs in UMC 65 nm technology capable to operate at a measured peak efficiency up to 3.08Gop/s/mW at 1.24 mW peak power, can achieve real-time phoneme extraction on a demanding RNN topology proposed in [1], consuming less than 13 mW of average power.

[5] G. E. Moore, "Progress in Digital Integrated Electronics," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 36–37, Sep. 2006, [TLDR] The complexity of integrated circuits is discussed, their manufacture, production, and deployment is identified, and trends to their future deployment are addressed.

[6] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974, [TLDR] This paper considers the design, fabrication, and characterization of very small Mosfet switching devices suitable for digital integrated circuits, using dimensions of the order of 1 /spl mu/.

[7] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 1–12.

[8] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, "TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, Mar. 2021.

[9] M. Abrash, "Creating the Future: Augmented Reality, the next Human-Machine Interface," in *2021 IEEE International Electron Devices Meeting (IEDM)*, Dec. 2021, pp. 1–11.

[10] S. Han, B. Liu, R. Cabezas, C. D. Twigg, P. Zhang, J. Petkau, T.-H. Yu, C.-J. Tai, M. Akbay, Z. Wang *et al.*, "MEgAtrack: Monochrome Egocentric Articulated Hand-Tracking for Virtual Reality," *ACM Transactions on Graphics*, vol. 39, no. 4, pp. 87:87:1–87:87:13, Aug. 2020.

[11] L. Chen, Y. Li, X. Bai, X. Wang, Y. Hu, M. Song, L. Xie, Y. Yan, and E. Yin, "Real-time Gaze Tracking with Head-eye Coordination for Head-mounted Displays," in *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, Oct. 2022, pp. 82–91.

[12] P. P. Ray, "A Review on TinyML: State-of-the-Art and Prospects," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, Apr. 2022.

[13] N. Tekin, A. Aris, A. Acar, S. Uluagac, and V. C. Gungor, "A Review of on-Device Machine Learning for IoT: An Energy Perspective," *Ad Hoc Networks*, vol. 153, p. 103348, Feb. 2024.

[14] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUnet: Tiny Deep Learning on IoT Devices," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., Dec. 2020, pp. 11 711–11 722.

[15] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[16] J. Choi, Z. Wang, S. Venkataramani, P. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized Clipping Activation for Quantized Neural Networks," *ArXiv*, Feb. 2018, [TLDR] It is shown, for the first time, that both weights and activations can be quantized to 4-bits of precision while still achieving accuracy comparable to full precision networks across a range of popular models and datasets.

[17] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022.

[18] S. Jain, A. Gural, M. Wu, and C. Dick, "Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 112–128, Mar. 2020.

[19] G. Rutishauser, F. Conti, and L. Benini, "Free Bits: Latency Optimization of Mixed-Precision Quantized Neural Networks on the Edge," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2023, pp. 1–5.

[20] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi, and L. Benini, "XpulpNN: Accelerating Quantized Neural Networks on RISC-V Processors Through ISA Extensions," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2020, pp. 186–191.

[21] R. Andri, G. Karunaratne, L. Cavigelli, and L. Benini, "ChewBaccaNN: A Flexible 223 TOPS/W BNN Accelerator," *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2021, [TLDR] ChewBaccaNN is presented, a 0.7 mm2 sized binary convolutional neural network (CNN) accelerator designed in GlobalFoundries 22 nm technology that can perform CIFAR-10 inference at 86.8% accuracy and perform inference on a binarized ResNet-18 trained with 8-bases Group-Net to achieve a 67.5% Top-1 accuracy.

[22] A. S. Waterman, "Design of the RISC-V Instruction Set Architecture," Ph.D. dissertation, University of California, Berkeley, United States – California, 2016.

[23] E. Cui, T. Li, and Q. Wei, "RISC-V Instruction Set Architecture Extensions: A Survey," *IEEE Access*, vol. 11, pp. 24 696–24 711, 2023.

[24] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, vol. 21, no. 9, pp. 948–960, Sep. 1972.

[25] K. Ueyoshi, I. A. Papistas, P. Houshmand, G. M. Sarda, V. Jain, M. Shi, Q. Zheng, S. Giraldo, P. Vrancx, J. Doevenspeck *et al.*, "DIANA: An End-to-End Energy-Efficient Digital and ANAlog Hybrid Neural Network SoC," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, Feb. 2022, pp. 1–3.

[26] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An Always-On 3.8 μJ/86% CIFAR-10 Mixed-Signal Binary CNN Processor With All Memory on Chip in 28-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan. 2019.

[27] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[28] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Jul. 2018, pp. 1–4.

[29] M. Eggimann, S. Mach, M. Magno, and L. Benini, "A RISC-V Based Open Hardware Platform for Always-On Wearable Smart Sensing," in *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*, Jun. 2019, pp. 169–174.

[30] A. S. Prasad, M. Scherer, F. Conti, D. Rossi, A. Di Mauro, M. Eggimann, J. T. Gómez, Z. Li, S. S. Sarwar, Z. Wang *et al.*, "Siracusa: A 16 nm Heterogenous RISC-V SoC for Extended Reality with At-MRAM Neural Engine," *ArXiv*, no. arXiv:2312.14750, Dec. 2023, accepted for publication at IEEE Journal of Solid-State Circuits.

[31] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, and Y. S. Shao, "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators," *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 554–566, Jun. 2021, [TLDR] CoSA leverages the regularities in DNN operators and hardware to formulate the DNN scheduling space into a mixed-integer programming (MIP) problem with algorithmic and architectural constraints, which can be solved to automatically generate a highly efficient schedule in one shot.

[32] M. Scherer, M. Magno, J. Erb, P. Mayer, M. Eggimann, and L. Benini, "TinyRadarNN: Combining Spatial and Temporal Convolutional Neural Networks for Embedded Gesture Recognition With Short Range Radars," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 336–10 346, Jul. 2021.

[33] M. Scherer, C. Cioflan, M. Magno, and L. Benini, "Work In Progress: Linear Transformers for TinyML," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2024.

[34] M. Scherer, G. Rutishauser, L. Cavigelli, and L. Benini, "CUTIE: Beyond PetaOp/s/W Ternary DNN Inference Acceleration With Better-Than-Binary Energy Efficiency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1020–1033, Apr. 2022.

[35] M. Scherer, A. D. Mauro, T. Fischer, G. Rutishauser, and L. Benini, "TCN-CUTIE: A 1,036-TOp/s/W, 2.72-µJ/Inference, 12.2-mW All-Digital Ternary Accelerator in 22-nm FDX Technology," *IEEE Micro*, vol. 43, no. 1, pp. 42–48, Jan. 2023.

[36] M. Scherer, L. Macan, V. Jung, P. Wiese, A. Burrello, F. Conti, and L. Benini, "Deeploy: Enabling Energy-Efficient Deployment of Small Language Models On Heterogeneous Microcontrollers," Mar. 2024, under Review at IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD).

[37] M. Scherer, K. Menachery, and M. Magno, "SmartAid: A Low-Power Smart Hearing Aid For Stutterers," in *2019 IEEE Sensors Applications Symposium (SAS)*, Mar. 2019, pp. 1–6.

[38] A. Di Mauro, M. Scherer, J. F. Mas, B. Bougenot, M. Magno, and L. Benini, "FlyDVS: An Event-Driven Wireless Ultra-Low Power Visual Sensor Node," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Feb. 2021, pp. 1851–1854.

[39] M. Scherer, P. Mayer, A. di Mauro, M. Magno, and L. Benini, "Towards Always-on Event-based Cameras for Long-lasting Battery-operated Smart Sensor Nodes," in *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, May 2021, pp. 1–6.

[40] A. Burrello, M. Scherer, M. Zanghieri, F. Conti, and L. Benini, "A Microcontroller is All You Need: Enabling Transformer Execution on Low-Power IoT Endnodes," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, Aug. 2021, pp. 1–6.

[41] A. Burrello, F. B. Morghet, M. Scherer, S. Benatti, L. Benini, E. Macii, M. Poncino, and D. J. Pagliari, "Bioformers: Embedding Transformers for Ultra-Low Power sEMG-based Gesture Recognition," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2022, pp. 1443–1448.

[42] A. Di Mauro, M. Scherer, D. Rossi, and L. Benini, "Kraken: A Direct Event/Frame-Based Multi-sensor Fusion SoC for Ultra-Efficient Visual Processing in Nano-UAVs," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, Aug. 2022, pp. 1–19.

[43] M. Scherer, A. Di Mauro, G. Rutishauser, T. Fischer, and L. Benini, "A 1036 TOp/s/W, 12.2 mW, 2.72 μJ/Inference All Digital TNN Accelerator in 22 nm FDX Technology for TinyML Applications," in *2022 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Apr. 2022, pp. 1–3.

[44] M. Scherer, F. Sidler, M. Rogenmoser, M. Magno, and L. Benini, "WideVision: A Low-Power, Multi-Protocol Wireless Vision

Platform for Distributed Surveillance," in *2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2022, pp. 394–399.

[45] P. Busia, A. Cossettini, T. M. Ingolfsson, S. Benatti, A. Burrello, M. Scherer, M. A. Scrugli, P. Meloni, and L. Benini, "EEGformer: Transformer-Based Epilepsy Detection on Raw EEG Traces for Low-Channel-Count Wearable Continuous Monitoring Devices," in *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Oct. 2022, pp. 640–644.

[46] G. Rutishauser, M. Scherer, T. Fischer, and L. Benini, "Ternarized TCN for μJ/Inference Gesture Recognition from DVS Event Frames," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 736–741.

[47] M. Scherer, M. Eggimann, A. D. Mauro, A. S. Prasad, F. Conti, D. Rossi, J. T. Gómez, Z. Li, S. S. Sarwar, Z. Wang *et al.*, "Siracusa: A Low-Power On-Sensor RISC-V SoC for Extended Reality Visual Processing in 16nm CMOS," in *ESSCIRC 2023- IEEE 49th European Solid State Circuits Conference (ESSCIRC)*, Sep. 2023, pp. 217–220.

[48] G. Rutishauser, M. Scherer, T. Fischer, and L. Benini, "7 μJ/Inference End-to-End Gesture Recognition from Dynamic Vision Sensor Data Using Ternarized Hybrid Convolutional Neural Networks," *Future Generation Computer Systems*, vol. 149, pp. 717–731, Dec. 2023.

[49] A. Mattei, M. Scherer, C. Cioflan, M. Magno, and L. Benini, "Securing Tiny Transformer-based Computer Vision Models: Evaluating Real-World Patch Attacks," in *9th World Forum on the Internet of Things (WF-IoT 2023)*, 2023.

[50] G. Islamoglu, M. Scherer, G. Paulin, T. Fischer, V. J. Jung, A. Garofalo, and L. Benini, "ITA: An Energy-Efficient Attention and Softmax Accelerator for Quantized Transformers," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2023, pp. 1–6.

[51] P. Busia, A. Cossettini, T. M. Ingolfsson, S. Benatti, A. Burrello, V. J. B. Jung, M. Scherer, M. A. Scrugli, A. Bernini, P. Ducouret *et al.*, "Reducing False Alarms in Wearable Seizure Detection with EEGformer: A Compact Transformer Model for MCUs," *IEEE Transactions on Biomedical Circuits and Systems*, pp. 1–13, 2024.

[52] V. J. B. Jung, A. Burrello, M. Scherer, F. Conti, and L. Benini, "Optimizing the Deployment of Tiny Transformers on Low-Power MCUs," *ArXiv*, no. arXiv:2404.02945, Apr. 2024, manuscript submitted for review at IEEE Transactions on Computers.

[53] V. Potocnik, A. D. Mauro, C. Leitner, M. Scherer, G. Rutishauser, L. Lamberti, and L. Benini, "Kraken: An Open-Source RISC-V SoC for Ultra-Low Power Multi-Modal Perception," *ArXiv*, Apr. 2024.

[54] F. Naujoks, Y. Forster, K. Wiedemann, and A. Neukum, "A Human-Machine Interface for Cooperative Highly Automated Driving," in *Advances in Human Aspects of Transportation*, N. A. Stanton, S. Landry, G. Di Bucchianico, and A. Vallicelli, Eds. Cham: Springer International Publishing, 2017, pp. 585–595.

[55] Q. Li, R. Gravina, Y. Li, S. H. Alsamhi, F. Sun, and G. Fortino, "Multi-User Activity Recognition: Challenges and Opportunities," *Information Fusion*, vol. 63, pp. 121–135, Nov. 2020.

[56] Y. Zhang, Y. Chen, H. Yu, X. Yang, and W. Lu, "Learning Effective Spatial–Temporal Features for sEMG Armband-Based Gesture Recognition," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6979–6992, Aug. 2020.

[57] J. Wu and R. Jafari, "Orientation Independent Activity/Gesture Recognition Using Wearable Motion Sensors," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1427–1437, Apr. 2019.

[58] R. Xu, S. Zhou, and W. J. Li, "MEMS Accelerometer Based Nonspecific-User Hand Gesture Recognition," *IEEE Sensors Journal*, vol. 12, no. 5, pp. 1166–1173, May 2012.

[59] A. Dementyev and J. A. Paradiso, "WristFlex: Low-Power Gesture Input with Wrist-Worn Pressure Sensors," in *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '14. New York, NY, USA: Association for Computing Machinery, Oct. 2014, pp. 161–166.

[60] A. Bandini and J. Zariffa, "Analysis of the Hands in Egocentric Vision: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 6, pp. 6846–6866, Jun. 2023.

[61] S. S. Rautaray and A. Agrawal, "Vision Based Hand Gesture Recognition for Human Computer Interaction: A Survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 1–54, Jan. 2015.

[62] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for Mobile Robot Navigation: Evaluation and Modeling," in *2015 International Conference on Advanced Robotics (ICAR)*. IEEE, Sep. 2015, pp. 388–394.

[63] Y. Zhang and C. Harrison, "Tomo: Wearable, Low-Cost Electrical Impedance Tomography for Hand Gesture Recognition," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, ser. UIST '15. New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 167–173.

[64] J. Yang, H. Zou, Y. Zhou, and L. Xie, "Learning Gestures From WiFi: A Siamese Recurrent Convolutional Architecture," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 763–10 772, Dec. 2019.

[65] J. Wang, L. Zhang, C. Wang, X. Ma, Q. Gao, and B. Lin, "Device-Free Human Gesture Recognition With Generative Adversarial Networks," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7678–7688, Aug. 2020.

[66] F. Adib, Z. Kabelac, D. Katabi, and R. C. Miller, "3D Tracking Via Body Radio Reflections," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. USA: USENIX Association, Apr. 2014, pp. 317–329.

[67] J. Lien, N. Gillian, M. E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev, "Soli: Ubiquitous Gesture Sensing with Millimeter Wave Radar," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 1–19, Jul. 2016, [TLDR] It is demonstrated that Soli can be used for robust gesture recognition and can track gestures with sub-millimeter accuracy, running at over 10,000 frames per second on embedded hardware.

[68] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A Robust and Energy-Efficient Classifier Using Brain-Inspired Hyperdimensional Computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 64–69.

[69] M. Magno, M. Pritz, P. Mayer, and L. Benini, "DeepEmote: Towards Multi-Layer Neural Networks in a Low Power Wearable Multi-Sensors Bracelet," in *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, Jun. 2017, pp. 32–37.

[70] C. Savaglio, P. Gerace, G. Di Fatta, and G. Fortino, "Data Mining at the IoT Edge," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, Jul. 2019, pp. 1–6.

[71] G. Fortino, S. Galzarano, R. Gravina, and W. Li, "A Framework for Collaborative Computing and Multi-Sensor Data Fusion in Body Sensor Networks," *Information Fusion*, vol. 22, pp. 50–70, Mar. 2015.

[72] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[73] F. Samie, L. Bauer, and J. Henkel, "From Cloud Down to Things: An Overview of Machine Learning in Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4921–4934, Jun. 2019.

[74] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, Aug. 2020, [TLDR] The former focuses on providing more optimal solutions to key problems in edge computing with the help of popular and effective AI technologies while the latter studies how to carry out the entire process of building AI models, i.e., model training and inference, on the edge.

[75] L. Cavigelli, P. Degen, and L. Benini, "CBinfer: Change-Based Inference for Convolutional Neural Networks on Video Data," *Proceedings of the 11th International Conference on Distributed Smart Cameras*, pp. 1–8, Sep. 2017, [TLDR] A novel algorithm is proposed and evaluated for change-based evaluation of CNNs for video data recorded with a static camera setting, exploiting the spatio-temporal sparsity of pixel changes to achieve an average speed-up of 8.6x over a cuDNN baseline on a realistic benchmark.

[76] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W Convolutional Network Accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2461–2475, Nov. 2017.

[77] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.

[78] L. Rabiner and B. Juang, "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, Jan. 1986.

[79] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[80] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*

(Applications and Reviews), vol. 37, no. 3, pp. 311–324, May 2007.

[81] A. K. H. Al-Saedi and A. H. H. Al-Asadi, "Survey of Hand Gesture Recognition Systems," *Journal of Physics: Conference Series*, vol. 1294, no. 4, p. 042003, Sep. 2019.

[82] S. Wang, J. Song, J. Lien, I. Poupyrev, and O. Hilliges, "Interacting with Soli: Exploring Fine-Grained Dynamic Gesture Recognition in the Radio-Frequency Spectrum," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ser. UIST '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 851–860.

[83] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," *ArXiv*, Mar. 2018, [TLDR] A systematic evaluation of generic convolutional and recurrent architectures for sequence modeling concludes that the common association between sequence modeling and recurrent networks should be reconsidered, and convolutionals should be regarded as a natural starting point for sequence modeled tasks.

[84] D. Jarrett, J. Yoon, and M. van der Schaar, "Dynamic Prediction in Clinical Survival Analysis Using Temporal Convolutional Networks," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 2, pp. 424–436, Feb. 2020.

[85] E. P. MatthewDavies and S. Bock, "Temporal Convolutional Networks for Musical Audio Beat Tracking," *2019 27th European Signal Processing Conference (EUSIPCO)*, pp. 1–5, Sep. 2019, [TLDR] Three highly promising attributes of TCNs for music analysis are demonstrated, namely: they achieve state-of-the-art performance on a wide range of existing beat tracking datasets, they are well suited to parallelisation and thus can be trained efficiently even on very large training data, and they require a small number of weights.

[86] S. Ahmed, K. D. Kallu, S. Ahmed, and S. H. Cho, "Hand Gestures Recognition Using Radar Sensors for Human-Computer-

Interaction: A Review," *Remote Sensing*, vol. 13, no. 3, p. 527, Jan. 2021.

[87] J. Wan, G. Guo, and S. Z. Li, "Explore Efficient Local Features from RGB-D Data for One-Shot Learning Gesture Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 8, pp. 1626–1639, Aug. 2016.

[88] D. Wu, L. Pigou, P.-J. Kindermans, N. D.-H. Le, L. Shao, J. Dambre, and J.-M. Odobez, "Deep Dynamic Neural Networks for Multimodal Gesture Segmentation and Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 8, pp. 1583–1597, Aug. 2016.

[89] A. D. Calin, "Gesture Recognition on Kinect Time Series Data Using Dynamic Time Warping and Hidden Markov Models," in *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Sep. 2016, pp. 264–271.

[90] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional Neural Networks for Human Action Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, Jan. 2013.

[91] O. Koller, N. C. Camgoz, H. Ney, and R. Bowden, "Weakly Supervised Learning with Multi-Stream CNN-LSTM-HMMs to Discover Sequential Parallelism in Sign Language Videos," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 9, pp. 2306–2320, Sep. 2020.

[92] B. Kellogg, V. Talla, and S. Gollakota, "Bringing Gesture Recognition to All Devices," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 303–316.

[93] S. Y. Kim, H. G. Han, J. W. Kim, S. Lee, and T. W. Kim, "A Hand Gesture Recognition Sensor Using Reflected Impulses," *IEEE Sensors Journal*, vol. 17, no. 10, pp. 2975–2976, May 2017.

[94] Y. Kim and B. Toomajian, "Hand Gesture Recognition Using Micro-Doppler Signatures With Convolutional Neural Net-

work," *IEEE Access*, vol. 4, pp. 7125–7130, 2016, [TLDR] The feasibility of recognizing human hand gestures using micro-Doppler signatures measured by Doppler radar with a deep convolutional neural network (DCNN) is investigated and the classification accuracy is found to be 85.6%.

[95] Y. Sun, T. Fei, S. Gao, and N. Pohl, "Automatic Radar-based Gesture Detection and Classification via a Region-based Deep Convolutional Neural Network," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 4300–4304.

[96] S. Hazra and A. Santra, "Radar Gesture Recognition System in Presence of Interference using Self-Attention Neural Network," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, Dec. 2019, pp. 1409–1414.

[97] J.-W. Choi, S.-J. Ryu, and J.-H. Kim, "Short-Range Radar Based Real-Time Hand Gesture Recognition Using LSTM Encoder," *IEEE Access*, vol. 7, pp. 33 610–33 618, 2019, [TLDR] A hand gesture recognition system for a real-time application of HCI using 60 GHz frequency-modulated continuous wave (FMCW) radar, Soli, developed by Google is proposed.

[98] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal Convolutional Networks for Action Segmentation and Detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Jul. 2017, pp. 1003–1012.

[99] F. Luo, S. Poslad, and E. Bodanese, "Temporal Convolutional Networks for Multiperson Activity Recognition Using a 2-D LIDAR," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7432–7442, Aug. 2020, [TLDR] This work clustered raw LIDAR data and classified the clusters into human and nonhuman classes in order to recognize humans in a scenario and built two neural networks, including a long short-term memory network and a temporal convolutional network (TCN) to classify trajectory samples into 15 activity classes collected from a kitchen.

[100] Y. Sun, T. Fei, F. Schliep, and N. Pohl, "Gesture Classification with Handcrafted Micro-Doppler Features using a FMCW Radar," in *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, Apr. 2018, pp. 1–4.

[101] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," in *Proceedings of the 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, 2016, pp. 125–125.

[102] A. Pandey and D. Wang, "TCNN: Temporal Convolutional Neural Network for Real-time Speech Enhancement in the Time Domain," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 6875–6879.

[103] T. S. Kim and A. Reiter, "Interpretable 3D Human Action Analysis with Temporal Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1623–1631, Jul. 2017, [TLDR] This work proposes to use a new class of models known as Temporal Convolutional Neural Networks (TCN) for 3D human action recognition, and aims to take a step towards a spatio-temporal model that is easier to understand, explain and interpret.

[104] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," *ArXiv*, no. arXiv:1801.06601, Jan. 2018.

[105] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, Jun. 2000.

[106] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[107] G. Bontempi, S. Ben Taieb, and Y.-A. Le Borgne, "Machine Learning Strategies for Time Series Forecasting," in *Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012, Tutorial Lectures*, ser. Lecture Notes in Business Information Processing, M.-A. Aufaure and E. Zimányi, Eds. Berlin, Heidelberg: Springer, 2013, pp. 62–77.

[108] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. ukasz Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[109] Y. Zhang, J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le, and Y. Wu, "Pushing the Limits of Semi-Supervised Learning for Automatic Speech Recognition," *ArXiv*, no. arXiv:2010.10504, Jul. 2022, comment: 11 pages, 3 figures, 5 tables. Accepted to NeurIPS SAS 2020 Workshop; v2: minor errors corrected.

[110] S. Zhao and B. Ma, "MossFormer: Pushing the Performance Limit of Monaural Speech Separation using Gated Single-Head Transformer with Convolution-Augmented Joint Self-Attentions," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Feb. 2023, comment: 5 pages, 3 figures, accepted by ICASSP 2023.

[111] A. Gazneli, G. Zimerman, T. Ridnik, G. Sharir, and A. Noy, "End-to-End Audio Strikes Back: Boosting Augmentations Towards An Efficient Audio Classification Network," *ArXiv*, no. arXiv:2204.11479, Jul. 2022.

[112] S. Kumar, P. Tiwari, and M. Zymbler, "Internet of Things Is a Revolutionary Approach for Future Technology Enhancement: A Review," *Journal of Big Data*, vol. 6, no. 1, p. 111, Dec. 2019.

[113] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention," in *Proceedings of the 37th International Confer-*

ence on Machine Learning, ser. ICML'20, vol. 119. JMLR.org, Jul. 2020, pp. 5156–5165.

[114] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The Efficient Transformer," in *International Conference on Learning Representations*, Sep. 2019.

[115] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-Attention with Linear Complexity," *ArXiv*, no. arXiv:2006.04768, Jun. 2020.

[116] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking Attention with Performers," in *International Conference on Learning Representations*, Feb. 2022.

[117] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv*, Apr. 2018, [TLDR] An audio dataset of spoken words designed to help train and evaluate keyword spotting systems and suggests a methodology for reproducible and comparable accuracy metrics for this task.

[118] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello Edge: Keyword Spotting on Microcontrollers," *ArXiv*, Nov. 2017, [TLDR] It is shown that it is possible to optimize these neural network architectures to fit within the memory and compute constraints of microcontrollers without sacrificing accuracy, and the depthwise separable convolutional neural network (DS-CNN) is explored and compared against other neural network architecture.

[119] P. M. Sørensen, B. Epp, and T. May, "A Depthwise Separable Convolutional Neural Network for Keyword Spotting on an Embedded System," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2020, no. 1, p. 10, Jun. 2020.

[120] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal Convolution for Real-Time Keyword Spotting on Mobile Devices," in *Interspeech 2019*. ISCA, Sep. 2019, pp. 3372–3376, [TLDR] A temporal convolution for real-time KWS on mobile devices that exploits temporal convolutions with a compact ResNet architecture and achieves more

than \textbf{385x} speedup on Google Pixel 1 and surpass the accuracy compared to the state-of-the-art model.

[121] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted Residual Learning for Efficient Keyword Spotting," in *Interspeech 2021*. ISCA, Aug. 2021, pp. 4538–4542, [TLDR] This work presents a broadcasted residual learning method to achieve high accuracy with small model size and computational load, and proposes a novel network architecture, Broadcasting-residual network (BC-ResNet), based on broadcasting residual learning and describes how to scale up the model according to the target device's resources.

[122] A. Berg, M. O'Connor, and M. T. Cruz, "Keyword Transformer: A Self-Attention Model for Keyword Spotting," in *Interspeech 2021*. ISCA, Aug. 2021, pp. 4249–4253, [TLDR] The Keyword Transformer (KWT), a fully self-attentional architecture that exceeds state-of-the-art performance across multiple tasks without any pre-training or additional data, is introduced.

[123] K. Ding, M. Zong, J. Li, and B. Li, "LETR: A Lightweight and Efficient Transformer for Keyword Spotting," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2022, pp. 7987–7991.

[124] R. Banner, Y. Nahshan, and D. Soudry, "Post Training 4-Bit Quantization of Convolutional Networks for Rapid-Deployment," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., Dec. 2019, pp. 7950–7958.

[125] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-Aware Automated Quantization With Mixed Precision," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8604–8612, Jun. 2019, [TLDR] The Hardware-Aware Automated Quantization (HAQ) framework is introduced which leverages the reinforcement learning to automatically determine the quantization policy, and takes the hardware accelerator's feedback in the design loop to generate direct feedback signals to the RL agent.

[126] Z. Liu, B. Oguz, A. Pappu, L. Xiao, S. Yih, M. Li, R. Krishnamoorthi, and Y. Mehdad, "BiT: Robustly Binarized Multi-Distilled Transformer," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 303–14 316, 2022.

[127] G. Prato, E. Charlaix, and M. Rezagholizadeh, "Fully Quantized Transformer for Machine Translation," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 1–14.

[128] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-BERT: Integer-only BERT Quantization," in *Proceedings of the 38th International Conference on Machine Learning*. PMLR, Jul. 2021, pp. 5506–5518.

[129] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, "BinaryBERT: Pushing the Limit of BERT Quantization," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Online: Association for Computational Linguistics, Aug. 2021, pp. 4334–4348.

[130] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: Quantized 8Bit BERT," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*. IEEE Computer Society, Dec. 2019, pp. 36–39.

[131] C. Zhao, T. Hua, Y. Shen, Q. Lou, and H. Jin, "Automatic Mixed-Precision Quantization Search of BERT," *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pp. 3427–3433, Aug. 2021, [TLDR] This paper proposes an automatic mixed-precision quantization framework designed for BERT that can conduct quantization and pruning simultaneously, and leverages Differentiable Neural Architecture Search to assign scale and precision for parameters in each sub-group

automatically, and at the same pruning out redundant groups of parameters.

[132] H. Lin, X. Cheng, X. Wu, and D. Shen, "CAT: Cross Attention in Vision Transformer," in *2022 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE Computer Society, Jul. 2022, pp. 1–6.

[133] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," *ArXiv*, no. arXiv:1606.08415, Jun. 2023, comment: Trimmed version of 2016 draft.

[134] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, Oct. 2020.

[135] X. Chen, S. Yin, D. Song, P. Ouyang, L. Liu, and S. Wei, "Small-Footprint Keyword Spotting with Graph Convolutional Network," *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 539–546, Dec. 2019, [TLDR] This study proposes a novel context-aware and compact architecture for keyword spotting task based on residual connection and bottleneck structure, and designs a compact and efficient network for KWS task.

[136] D. Ng, Y. Chen, B. Tian, Q. Fu, and E. S. Chng, "Convmixer: Feature Interactive Convolution with Curriculum Learning for Small Footprint and Noisy Far-Field Keyword Spotting," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2022, pp. 3603–3607.

[137] Y. Gong, Y.-A. Chung, and J. Glass, "AST: Audio Spectrogram Transformer," *Interspeech 2021*, pp. 571–575, Aug. 2021, [TLDR] The Audio Spectrogram Transformer is introduced, the first convolution-free, purely attention-based model for audio classification, which achieves new state-of-the-art results on various audio classification benchmarks.

[138] D. C. de Andrade, S. Leo, M. Viana, and C. Bernkopf, "A Neural Attention Model for Speech Command Recognition," *ArXiv*, Aug. 2018, [TLDR] A convolutional recurrent network with attention for speech command recognition that establishes a new state-of-the-art accuracy of 94.1% and allows inspecting what regions of the audio were taken into consideration by the network when outputting a given category.

[139] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming Keyword Spotting on Mobile Devices," *Interspeech 2020*, pp. 2277–2281, Oct. 2020, [TLDR] A Tensorflow/Keras based library is designed which allows automatic conversion of non-streaming models to streaming ones with minimum effort and also explores novel KWS models with multi-head attention which reduce the classification error over the state-of-art by 10% on Google speech commands data sets V2.

[140] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and Benchmarking of Machine Learning Accelerators," *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–9, Sep. 2019, [TLDR] This paper surveys the current state of processors and accelerators that have been publicly announced with performance and power consumption numbers, and selects and benchmark two commercially available low size, weight, and power (SWaP) accelerators as these processors are the most interesting for embedded and mobile machine learning inference applications that are most applicable to the DoD and other SWaP constrained users.

[141] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning Both Weights and Connections for Efficient Neural Networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, Dec. 2015, pp. 1135–1143.

[142] F. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. Dally, and K. Keutzer, "SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <1MB Model Size," *ArXiv*, Feb. 2016, [TLDR] This work proposes a small DNN architecture called SqueezeNet, which achieves AlexNet-level accuracy on

ImageNet with 50x fewer parameters and is able to compress to less than 0.5MB (510x smaller than AlexNet).

[143] L. Cavigelli, G. Rutishauser, and L. Benini, "EBPC: Extended Bit-Plane Compression for Deep Neural Network Inference and Training Accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 723–734, Dec. 2019, [TLDR] This work introduces and evaluates a novel, hardware-friendly, and lossless compression scheme for the feature maps present within convolutional neural networks, and achieves compression factors for gradient map compression during training that are even better than for inference.

[144] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with Binary Weights During Propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, Dec. 2015, pp. 3123–3131.

[145] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[146] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., Dec. 2016, pp. 4114–4122.

[147] B. Liu, F. Li, X. Wang, B. Zhang, and J. Yan, "Ternary Weight Networks," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Jun. 2023, pp. 1–5.

[148] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained Ternary Quantization," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[149] X. Lin, C. Zhao, and W. Pan, "Towards Accurate Binary Convolutional Neural Network," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 344–352.

[150] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Petrot, "Ternary Neural Networks for Resource-Efficient AI Applications," *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2547–2554, May 2017, [TLDR] This paper proposes ternary neural networks (TNNs) in order to make deep learning more resource-efficient, and designs a purpose-built hardware architecture for TNNs and implements it on FPGA and ASIC.

[151] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary Neural Networks: A Survey," *Pattern Recognition*, vol. 105, p. 107281, Sep. 2020.

[152] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: Wide Reduced-Precision Networks," in *International Conference on Learning Representations*, Feb. 2018.

[153] J. Choi, P. I.-J. Chuang, Z. Wang, S. Venkataramani, V. Srinivasan, and K. Gopalakrishnan, "Bridging the Accuracy Gap for 2-bit Quantized Neural Networks (QNN)," *ArXiv*, no. arXiv:1807.06964, Jul. 2018, comment: arXiv admin note: substantial text overlap with arXiv:1805.06085.

[154] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2016, pp. 236–241.

[155] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst, "BinarEye: An Always-on Energy-Accuracy-Scalable Binary CNN Processor with All Memory on Chip in 28nm Cmos," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*, Apr. 2018, pp. 1–4.

[156] S. Jain, S. K. Gupta, and A. Raghunathan, "TiM-DNN: Ternary In-Memory Accelerator for Deep Neural Networks,"

IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 7, pp. 1567–1577, Jul. 2020, [TLDR] TIM-DNN, a programmable in-memory accelerator that is specifically designed to execute ternary DNNs, is proposed and evaluated across a suite of state-of-the-art DNN benchmarks including both deep convolutional and recurrent neural networks.

[157] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-Tile 2.4-Mb In-Memory-Computing CNN Accelerator Employing Charge-Domain Compute," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, Jun. 2019, [TLDR] This paper addresses data movement via an in-memory-computing accelerator that employs charged-domain mixed-signal operation for enhancing compute SNR and, thus, scalability in large-scale matrix-vector multiplications.

[158] M. Klachko, M. R. Mahmoodi, and D. B. Strukov, "Improving Noise Tolerance of Mixed-Signal Neural Networks," in *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*. IEEE, 2019, pp. 1–8.

[159] P. C. Knag, G. K. Chen, H. E. Sumbul, R. Kumar, S. K. Hsu, A. Agarwal, M. Kar, S. Kim, M. A. Anders, H. Kaul *et al.*, "A 617-TOPS/W All-Digital Binary Neural Network Accelerator in 10-nm FinFET CMOS," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1082–1092, Apr. 2021, [TLDR] The bit-serial binary operation allows for bit-accurate operation and high DNN accuracy that multibit analog compute-in-memory designs struggle to attain and provides favorable energy tradeoffs compared with small-integer digital DNN accelerators.

[160] Q. Hu, P. Wang, and J. Cheng, "From Hashing to CNNs: Training Binary Weight Networks via Hashing," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 3247–3254.

[161] G. Cerutti, R. Andri, L. Cavigelli, E. Farella, M. Magno, and L. Benini, "Sound Event Detection with Binary Neural Networks on Tightly Power-Constrained IoT Devices," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '20. New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 19–24.

[162] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "GXNOR-Net: Training Deep Neural Networks with Ternary Weights and Activations Without Full-Precision Memory Under a Unified Discretization Framework," *Neural Networks*, vol. 100, pp. 49–58, 2018.

[163] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[164] A. Bulat and G. Tzimiropoulos, "XNOR-Net++: Improved Binary Neural Networks," in *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*. BMVA Press, 2019, p. 62.

[165] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *ArXiv*, Jun. 2016, [TLDR] DoReFa-Net, a method to train convolutional neural networks that have low bitwidth weights and activations using low bit width parameter gradients, is proposed and can achieve comparable prediction accuracy as 32-bit counterparts.

[166] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Structured Binary Neural Networks for Accurate Image Classification and Semantic Segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Jun. 2019, pp. 413–422.

[167] H. Phan, D. Huynh, Y. He, M. Savvides, and Z. Shen, "MoBiNet: A Mobile Binary Network for Image Classification," *2020 IEEE Winter Conference on Applications of Computer Vision*

(WACV), pp. 3442–3451, Mar. 2020, [TLDR] A novel neural network architecture, namely MoBi- Net - Mobile Binary Network in which skip connections are manipulated to prevent information loss and vanishing gradient, thus facilitate the training process and results in an effectively small model while keeping the accuracy comparable to existing ones.

[168] A. Byerly, T. Kalganova, and I. Dear, "A Branching and Merging Convolutional Network with Homogeneous Filter Capsules," *ArXiv*, Jan. 2020, [TLDR] A convolutional neural network design with additional branches after certain convolutions so that it can extract features with differing effective receptive fields and levels of abstraction establishes a new state of the art for the MNIST dataset with an accuracy of 99.84%.

[169] X. Sun, S. Yin, X. Peng, R. Liu, J.-s. Seo, and S. Yu, "XnorRRAM: A Scalable and Parallel Resistive Synaptic Architecture for Binary Neural Networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2018, pp. 1423–1428.

[170] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big Transfer (BiT): General Visual Representation Learning," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 491–507.

[171] P. Yin, S. Zhang, J. Lyu, S. Osher, Y. Qi, and J. Xin, "BinaryRelax: A Relaxation Approach for Training Deep Neural Networks with Quantized Weights," *SIAM Journal on Imaging Sciences*, vol. 11, no. 4, pp. 2205–2223, Jan. 2018, [TLDR] BinaryRelax is proposed, a simple two-phase algorithm for training deep neural networks with quantized weights that relax the hard constraint into a continuous regularizer via Moreau envelope, which turns out to be the squared Euclidean distance to the set of quantization weights.

[172] S. Darabi, M. Belbahri, M. Courbariaux, and V. Nia, "BNN+: Improved Binary Network Training," *ArXiv*, Sep. 2018, [TLDR] An improved binary training method is proposed, by intro-

ducing a new regularization function that encourages training weights around binary values and introducing an improved approximation of the derivative of the $sign$ activation function in the backward computation.

[173] H. Touvron, A. Vedaldi, M. Douze, and H. J'egou, "Fixing the Train-Test Resolution Discrepancy: FixEfficientNet," *ArXiv*, Mar. 2020, [TLDR] This strategy is advantageously combined with recent training recipes from the literature and significantly outperforms the initial architecture with the same number of parameters, and establishes the new state of the art for ImageNet with a single crop.

[174] L. Cavigelli and L. Benini, "RPR: Random Partition Relaxation for Training; Binary and Ternary Weight Neural Networks," *ArXiv*, Jan. 2020, [TLDR] Random Partition Relaxation (RPR) is presented, a method for strong quantization of neural networks weight to binary (+1/-1) and ternary (+1/0/1) values and an SGD-based training method that can be integrated into existing frameworks.

[175] M. Spallanzani, L. Cavigelli, G. P. Leonardi, M. Bertogna, and L. Benini, "Additive Noise Annealing and Approximation Properties of Quantized Neural Networks," *ArXiv*, May 2019, [TLDR] A novel gradient-based training algorithm for quantized neural networks that generalizes the straight-through estimator, acting on noise applied to the network's parameters, showing state-of-the-art performance on AlexNet and MobileNetV2 for ternary networks.

[176] J. Faraone, N. Fraser, G. Gambardella, M. Blott, and P. H. W. Leong, "Compressing Low Precision Deep Neural Networks Using Sparsity-Induced Regularization in Ternary Networks," in *Neural Information Processing*, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, Eds. Cham: Springer International Publishing, 2017, pp. 393–404.

[177] A. Marban, D. Becking, S. Wiedemann, and W. Samek, "Learning Sparse & Ternary Neural Networks with Entropy-Constrained Trained Ternarization (EC2T)," *2020 IEEE/CVF*

Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 3105–3113, Jun. 2020, [TLDR] Entropy-Constrained Trained Ternarization (EC2T), a general framework to create sparse and ternary neural networks which are efficient in terms of storage and computation, is proposed and validated in CIFAR-10, CIFar-100, and ImageNet datasets.

[178] R. Ding, T.-W. Chin, Z. Liu, and D. Marculescu, "Regularizing Activation Distribution for Training Binarized Deep Networks," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11 400–11 409, Jun. 2019, [TLDR] The experiments show that the distribution loss can consistently improve the accuracy of BNNs without losing their energy benefits and equipped with the proposed regularization, BNN training is shown to be robust to the selection of hyper-parameters including optimizer and learning rate.

[179] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[180] L. Cavigelli, M. Magno, and L. Benini, "Accelerating Real-Time Embedded Scene Labeling with Convolutional Networks," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, Jun. 2015, pp. 1–6.

[181] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A Survey of Accelerator Architectures for Deep Neural Networks," *Engineering*, vol. 6, no. 3, pp. 264–274, Mar. 2020.

[182] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2018, pp. 1–14.

[183] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W Precision-Scalable Processor for Real-Time Large-Scale Convnets," *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, pp. 1–2, Jun. 2016, [TLDR] A low-power precision-scalable processor for ConvNets or convolutional neural networks (CNN) is imple-

mented in a 40nm technology and is the first to both exploit the sparsity of convolutions and to implement dynamic precision-scalability enabling supply- and energy scaling.

[184] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Hyperdrive: A Multi-Chip Systolically Scalable Binary-Weight CNN Inference Engine," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 309–322, Jun. 2019.

[185] A. D. Mauro, F. Conti, P. D. Schiavone, D. Rossi, and L. Benini, "Always-On 674μW@4GOP/s Error Resilient Binary Neural Networks With Aggressive SRAM Voltage Scaling on a 22-nm IoT End-Node," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 11, pp. 3905–3918, Nov. 2020.

[186] S. Okumura, M. Yabuuchi, K. Hijioka, and K. Nose, "A Ternary Based Bit Scalable, 8.80 TOPS/W CNN accelerator with Many-core Processing-in-memory Architecture with 896K synapses/mm2," in *2019 Symposium on VLSI Technology*, Jun. 2019, pp. C248–C249.

[187] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, T. Kuroda *et al.*, "BRein Memory: A 13-Layer 4.2 K Neuron/0.8 M Synapse Binary/Ternary Reconfigurable in-Memory Deep Neural Network Accelerator in 65 nm CMOS," *2017 Symposium on VLSI Circuits*, pp. C24–C25, Jun. 2017, [TLDR] A versatile reconfigurable accelerator for binary/ternary deep neural networks (DNNs) is presented, which features a massively parallel in-memory processing architecture and stores varieties of binary/ Sternary DNNs with a maximum of 13 layers, 4.2 K neurons, and 0.8 M synapses on chip.

[188] A. Ardakani, Z. Ji, S. C. Smithson, B. Meyer, and W. Gross, "Learning Recurrent Binary/Ternary Weights," *ArXiv*, Sep. 2018, [TLDR] A method that can learn binary and ternary weights during the training phase to facilitate hardware implementations of RNNs is introduced, which replaces all multiply-accumulate operations by simple accumulations, bringing signif-

icant benefits to custom hardware in terms of silicon area and power consumption.

[189] S. Sen and A. Raghunathan, "Approximate Computing for Long Short Term Memory (LSTM) Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2266–2276, Nov. 2018.

[190] X. Zhou, Z. Du, S. Zhang, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Addressing Sparsity in Deep Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1858–1871, Oct. 2019.

[191] Z. Yuan, Y. Liu, J. Yue, Y. Yang, J. Wang, X. Feng, J. Zhao, X. Li, and H. Yang, "STICKER: An Energy-Efficient Multi-Sparsity Compatible Accelerator for Convolutional Neural Networks in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 2, pp. 465–477, Feb. 2020, [TLDR] Three new features are proposed in this article to support wide sparsity distribution efficiently and include a multi-sparsity-compatible set-associative convolution processing element (PE) array, designed to efficiently carry out convolution operations under different sparsity modes.

[192] O. Muller, A. Prost-Boucle, A. Bourge, and F. Pétrot, "Efficient Decompression of Binary Encoded Balanced Ternary Sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1962–1966, Aug. 2019.

[193] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow, "Thermometer Encoding: One Hot Way To Resist Adversarial Examples," in *International Conference on Learning Representations*, Feb. 2018.

[194] K. Goetschalckx and M. Verhelst, "Breaking High-Resolution CNN Bandwidth Barriers With Enhanced Depth-First Execution," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 323–331, Jun. 2019.

[195] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural Language Processing (Almost) from Scratch," *Journal of Machine Learning Research*, 2011.

[196] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, "Quentin: An Ultra-Low-Power PULPissimo SoC in 22nm FDX," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, Oct. 2018, pp. 1–3.

[197] A. Pullini, D. Rossi, G. Haugou, and L. Benini, "μDMA: An autonomous I/O subsystem for IoT end-nodes," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sep. 2017, pp. 1–8.

[198] J. S. P. Giraldo, V. Jain, and M. Verhelst, "Efficient Execution of Temporal Convolutional Networks for Embedded Keyword Spotting," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 12, pp. 2220–2228, Dec. 2021.

[199] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A Low Power, Fully Event-Based Gesture Recognition System," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 7388–7397.

[200] E. Ceolini, C. Frenkel, S. B. Shrestha, G. Taverni, L. Khacef, M. Payvand, and E. Donati, "Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing," *Frontiers in Neuroscience*, vol. 14, 2020.

[201] T. M. Ingolfsson, M. Hersche, X. Wang, N. Kobayashi, L. Cavigelli, and L. Benini, "EEG-TCNet: An Accurate Temporal Convolutional Network for Embedded Motor-Imagery Brain–Machine Interfaces," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2020, pp. 2958–2965.

[202] M. Zanghieri, S. Benatti, A. Burrello, V. J. Kartsch Morinigo, R. Meattini, G. Palli, C. Melchiorri, and L. Benini, "sEMG-based Regression of Hand Kinematics with Temporal Convolu-

tional Networks on a Low-Power Edge Microcontroller," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, Aug. 2021, pp. 1–6.

[203] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open Foundation and Fine-Tuned Chat Models," *ArXiv*, no. arXiv:2307.09288, Jul. 2023.

[204] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth *et al.*, "Gemini: A Family of Highly Capable Multimodal Models," *ArXiv*, no. arXiv:2312.11805, Dec. 2023.

[205] Y. Chen, K. Ren, K. Song, Y. Wang, Y. Wang, D. Li, and L. Qiu, "EEGFormer: Towards Transferable and Interpretable Large-Scale EEG Foundation Model," *ArXiv*, no. arXiv:2401.10278, Jan. 2024, comment: A preprint version of an ongoing work.

[206] C. Wang, V. Subramaniam, A. U. Yaari, G. Kreiman, B. Katz, I. Cases, and A. Barbu, "BrainBERT: Self-Supervised Representation Learning for Intracranial Recordings," in *The Eleventh International Conference on Learning Representations*, Sep. 2022.

[207] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog *et al.*, "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances," in *Conference on Robot Learning*, Apr. 2022, [TLDR] This work proposes to provide real-world grounding by means of pretrained skills, which are used to constrain the model to propose natural language actions that are both feasible and contextually appropriate, and shows how low-level skills can be combined with large language models so that the language model provides high-level knowledge about the procedures for performing complex and temporally extended instructions.

[208] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu *et al.*, "PaLM-E: An Embodied Multimodal Language Model," in *Proceedings of the 40th International Conference on Machine*

Learning, ser. ICML'23, vol. 202. JMLR.org, Jul. 2023, pp. 8469–8488.

[209] R. Eldan and Y. Li, "TinyStories: How Small Can Language Models Be and Still Speak Coherent English?" *ArXiv*, no. arXiv:2305.07759, May 2023.

[210] P. Zhang, G. Zeng, T. Wang, and W. Lu, "TinyLlama: An Open-Source Small Language Model," *ArXiv*, no. arXiv:2401.02385, Jan. 2024, comment: Technical Report.

[211] Y. LeCun, "Deep Learning Hardware: Past, Present, and Future," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, Feb. 2019, pp. 12–19.

[212] J. Choquette, "NVIDIA Hopper H100 GPU: Scaling Performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, May 2023.

[213] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Feb. 2021, pp. 2–14.

[214] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze *et al.*, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, Oct. 2018, pp. 579–594.

[215] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of Experts," *ArXiv*, no. arXiv:2401.04088, Jan. 2024, comment: See more details at https://mistral.ai/news/mixtral-of-experts/.

[216] R. Firoozi, J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman *et al.*, "Foundation Models in Robotics: Applications, Challenges, and the Future," *ArXiv*, no. arXiv:2312.07843, Dec. 2023.

[217] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction," in *International Conference on Learning Representations*, Oct. 2020.

[218] E. Frantar and D. Alistarh, "Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning," in *Advances in Neural Information Processing Systems*, Oct. 2022.

[219] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models," in *Proceedings of the 40th International Conference on Machine Learning*. PMLR, Jul. 2023, pp. 38 087–38 099.

[220] Y. Jeon, C. Lee, K. Park, and H.-y. Kim, "A Frustratingly Easy Post-Training Quantization Scheme for LLMs," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14 446–14 461.

[221] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang *et al.*, "Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Oct. 2019, pp. 1314–1324.

[222] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Memory-efficient Patch-based Inference for Tiny Deep Learning," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 2346–2358.

[223] F. Svoboda, J. Fernandez-Marques, E. Liberis, and N. D. Lane, "Deep Learning on Microcontrollers: A Study on Deployment Costs and Challenges," in *Proceedings of the 2nd European Workshop on Machine Learning and Systems*, ser. EuroMLSys '22. New York, NY, USA: Association for Computing Machinery, Apr. 2022, pp. 54–63.

[224] S. S. Saha, S. S. Sandha, and M. Srivastava, "Machine Learning for Microcontroller-Class Hardware: A Review," *IEEE Sensors Journal*, vol. 22, no. 22, pp. 21 362–21 390, Nov. 2022.

[225] M. Maas, U. Beaugnon, A. Chauhan, and B. Ilbeyi, "TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, Dec. 2022, pp. 123–137.

[226] M. D. Moffitt, "MiniMalloc: A Lightweight Memory Allocator for Hardware-Accelerated Machine Learning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ser. ASPLOS '23. New York, NY, USA: Association for Computing Machinery, Feb. 2024, pp. 238–252.

[227] M. Spallanzani, G. Rutishauser, M. Scherer, A. Burrello, F. Conti, and L. Benini, "QuantLab: A Modular Framework for Training and Deploying Mixed-Precision NNs," TinyML Summit, 2022.

[228] F. Angiolini, L. Benini, and A. Caprara, "An Efficient Profile-based Algorithm for Scratchpad Memory Partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1660–1676, Nov. 2005.

[229] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "TetriSched: Global Rescheduling with Adaptive Plan-Ahead in Dynamic Heterogeneous Clusters," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: Association for Computing Machinery, Apr. 2016, pp. 1–16.

[230] "Mako Templates for Python," https://github.com/sqlalchemy/mako/releases.

[231] B. Zhang and R. Sennrich, "Root Mean Square Layer Normalization," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[232] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "HellaSwag: Can a Machine Really Finish Your Sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 4791–4800, [TLDR] The construction of HellaSwag, a new challenge dataset, and its resulting difficulty, sheds light on the inner workings of deep pretrained models, and suggests a new path forward for NLP research, in which benchmarks co-evolve with the evolving state-of-the-art in an adversarial way, so as to present ever-harder challenges.

[233] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 2381–2391, [TLDR] A new kind of question answering dataset, OpenBookQA, modeled after open book exams for assessing human understanding of a subject, and oracle experiments designed to circumvent the knowledge retrieval bottleneck demonstrate the value of both the open book and additional facts.

[234] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "WinoGrande: An Adversarial Winograd Schema Challenge at Scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, Aug. 2021.

[235] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge," *ArXiv*, no. arXiv:1803.05457, Mar. 2018.

[236] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "BoolQ: Exploring the Surprising Diffi-

culty of Natural Yes/No Questions," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 2924–2936.

[237] Y. Bisk, R. Zellers, R. Le Bras, J. Gao, and Y. Choi, "PIQA: Reasoning about Physical Commonsense in Natural Language," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 7432–7439, Apr. 2020, [TLDR] The task of physical commonsense reasoning and a corresponding benchmark dataset Physical Interaction: Question Answering or PIQA are introduced and analysis about the dimensions of knowledge that existing models lack are provided, which offers significant opportunities for future research.

[238] L. Sutawika, L. Gao, H. Schoelkopf, S. Biderman, J. Tow, B. Abbasi, b. fattori, C. Lovering, farzanehnakhaee70, J. Phang *et al.*, "A Framework for Few-shot Language Model Evaluation," Zenodo, Dec. 2023.

[239] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang *et al.*, "GPT-NeoX-20B: An Open-Source Autoregressive Language Model," in *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, A. Fan, S. Ilic, T. Wolf, and M. Gallé, Eds. virtual+Dublin: Association for Computational Linguistics, May 2022, pp. 95–136.

[240] Z. Liu, C. Zhao, F. Iandola, C. Lai, Y. Tian, I. Fedorov, Y. Xiong, E. Chang, Y. Shi, R. Krishnamoorthi *et al.*, "MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases," *ArXiv*, no. arXiv:2402.14905, Feb. 2024.

[241] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau *et al.*, "MLPerf Tiny Benchmark," Aug. 2021.

[242] E. Ruedas, "LLM Pipelines: Seamless Integration on Embedded Devices," Virtual Event, Mar. 2024.

[243] X. Chu, L. Qiao, X. Lin, S. Xu, Y. Yang, Y. Hu, F. Wei, X. Zhang, B. Zhang, X. Wei *et al.*, "MobileVLM : A Fast, Strong and Open Vision Language Assistant for Mobile Devices," *ArXiv*, no. arXiv:2312.16886, Dec. 2023, comment: Tech Report.

[244] A. Karpathy, "Llama2.c," Mar. 2024.

[245] Joey (e/λ) [@shxf0072], "@karpathy llama2.c running on galaxy watch 4 https://t.co/sMPCZM3WE4," Dec. 2023.

[246] iFixit, "Samsung Galaxy Watch4 and Watch4 Classic Teardown," https://url.zip/4befa14, Sep. 2021.

[247] A. Frumusanu, "The Snapdragon 888 vs The Exynos 2100: Cortex-X1 & 5nm - Who Does It Better?" https://www.anandtech.com/show/16463/snapdragon-888-vs-exynos-2100-galaxy-s21-ultra, Feb. 2021.

Curriculum Vitae

Moritz Scherer was born on August 1, 1996 and grew up in Germany and Switzerland. He received his BSc and MSc in "Electrical Engineering and Information Technology" from ETH Zurich, Switzerland in 2018 and 2020, respectively. In 2020, he joined the Integrated Systems Laboratory of ETH Zurich as a PhD candidate under the supervision of Prof. Dr. Luca Benini. His research interests include algorithm design, computer architecture, and hardware acceleration of deep learning applications.