IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 October 2024, accepted 31 October 2024, date of publication 14 November 2024, date of current version 26 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3498462

APPLIED RESEARCH

Polara-Keras2c: Supporting Vectorized AI Models on RISC-V Edge Devices

NIZAR EL ZARIF, MOHAMMADHOSSEIN ASKARI HEMMAT, THEO DUPUIS, JEAN-PIERRE DAVID, AND YVON SAVARIA, (Life Fellow, IEEE)
Department of Electrical Engineering, Polytechnique Montréal, Montreal, QC H3T 1J4, Canada
Corresponding author: Nizar El Zarif (nizar.el-zarif@polymtl.ca)

This work was supported by CMC.

ABSTRACT The rise of edge computing has introduced unique challenges for deploying efficient AI solutions in resource-limited environments. While traditional AI frameworks are powerful, they often fall short in meeting the requirements of edge computing, such as low latency, constrained computational power, and energy efficiency. This paper presents Polara-Keras2c, an optimized evolution of Keras2c designed specifically for edge computing. Polara-Keras2c enhances compatibility with bare-metal systems, incorporates RISC-V vector extension optimization, and is customized for the Polara architecture. By converting pre-trained Keras models into optimized C code for bare-metal execution on edge devices, Polara-Keras2c enables advanced AI models to operate efficiently in resource-constrained environments. The framework supports fixed-point arithmetic, achieving a minimal accuracy impact of only 0.03% when tested on the MNIST dataset, and offers a streamlined approach for rapid prototyping. Experimental results reveal that Polara-Keras2c achieves up to 4.81 times faster convolution processing with a 64×64 input size compared to scalar processing, significantly enhancing computational efficiency and reducing energy consumption. These capabilities position Polara-Keras2c as a transformative tool in real-time, energy-efficient AI processing for edge devices, pushing forward the evolution of edge computing.

INDEX TERMS Vector processor, artificial intelligence, bare metal, real-time, edge computing, embedded systems.

I. INTRODUCTION

Artificial Intelligence (AI) has witnessed remarkable expansion, marked by significant advances in neural network techniques and architectures for diverse applications. However, integrating AI into edge computing, characterized by data processing close to the data source, presents unique challenges. Combining AI with edge computing leads to intelligent systems capable of real-time decision-making, but they often grapple with hardware limitations, software incompatibilities, and energy constraints. Polara-Keras2c, the focus of this paper, addresses these challenges by optimizing AI model deployment for resource-constrained edge devices.

A key inflection point of AI growth was the unveiling of Alexnet [1], a moment that marked a significant leap in the capabilities of neural networks. Since then, the evolution of AI has been characterized by significant advances in neural network techniques and architectures for various applications [2]. This includes developing gated recurrent units (GRUs) [3], which have become a mainstay in natural language processing tasks due to their efficiency in handling sequential data like text and speech. Another innovation is variational autoencoders (VAEs) [4], which have shown great promise in generative tasks such as image generation and anomaly detection, offering a novel way of learning complex data distributions. Lastly, generative adversarial networks (GANs) [5] have revolutionized the field with their ability to generate highly realistic images and videos, finding applications in areas ranging from art creation to data augmentation and style transfer. Each architectural innovation has opened new possibilities and applications, significantly expanding the scope and impact of AI technologies.

Edge computing, characterized by data processing close to the source rather than on distant cloud servers, has only

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

intensified AI adoption. Intelligent systems combining AI with edge computing are effective for real-time decision-making using local data in which latency, bandwidth, and data privacy are paramount, such as autonomous vehicles, urban planning, and healthcare. The proliferation of AI in edge computing stems from hardware, software, and algorithmic advancements, facilitating optimal data processing even on devices with stringent resource limits. This convergence promises transformative applications across industry, agriculture, energy, housing, healthcare, and environmental monitoring [6].

However, using AI in microcontroller-based systems is not without challenges. Constrained computing and memory capacities limit the complexity of AI solutions these microcontrollers can host. The field also grapples with standardization issues, as microcontrollers might employ disparate AI frameworks or languages, complicating cross-device model deployment. Power consumption, vital for devices like wearables and remote sensors, is another concern, given AI's intensive processing demands. Past solutions, such as Quantization, pruning, distillation, and binarization, aimed to facilitate AI deployment issues on microcontrollers [7]. Yet, the problem of model generalization remains, as models trained on one dataset might falter on another due to concealed biases in diverse data collection methodologies [8].

A. MOTIVATION AND CONTRIBUTIONS

The shift towards edge computing and processing data near its source brings benefits like reduced latency, enhanced reliability, and improved privacy. However, deploying AI solutions on edge devices poses significant challenges due to processing power, memory, and energy constraints. Often designed for more robust environments, traditional AI frameworks struggle in these settings. Polara-Keras2c emerges as a solution, streamlining the deployment of AI models on edge devices by leveraging the hardware's full capabilities without an Operating System (OS) overhead. It is optimized for various applications and devices, making it a versatile and powerful tool for edge AI. Notably, Polara-Keras2c offers distinct advantages over traditional machine-learning environments by prioritizing bare metal support, optimizing for RISC-V vector extensions, and enhancing rapid prototyping, especially when integrated with custom hardware architectures like those proposed in the Polara project [9]. Polara represents a code name for an advanced research initiative focused on enhancing the open-source Ara vector processor, as outlined in [10]. This project aims to integrate bit-serial and other advanced vector instructions into the processor. Notable advances in this domain are evident in projects like Quark [11] and Sparq [12]. Both Quark and Sparq eschew floating-point vector units to conserve area. Moreover, ongoing developments include complete multi-core versions of the core, equipped with fully enabled floating-point vector capabilities.

The growing demand for real-time AI applications in healthcare, autonomous vehicles, and industrial automation sectors has made edge computing increasingly important. The ability to process data locally and make immediate decisions is invaluable in these sectors. However, existing AI frameworks are typically designed for environments with abundant computing resources and fail to account for the unique constraints of edge devices. Recognizing this gap, Polara-Keras2c was developed to bring the power of AI to edge computing. The framework supports deploying AI models in resource-constrained environments and optimizes their performance by leveraging the unique features of the RISC-V architecture. Its capability to run AI models on bare metal systems without the overhead of an OS represents a significant advancement in edge computing. Furthermore, Polara-Keras2c's design facilitates rapid prototyping and integration with custom hardware platforms, making it an adaptable and powerful tool for a wide range of edge AI applications.

AI frameworks and benchmarks tailored for bare metal operation could circumvent these challenges [13]. By eliminating the need for an OS, such a framework can harness the hardware's full capabilities, resulting in superior performance and energy efficiency. Furthermore, its lightweight and adaptable nature can be optimized for various applications and devices.

Conversely, Keras2c is a proposed framework for converting Keras deep learning models into simple static C code [14]. It includes an assortment of library files and supports a range of vital layers, including dense, convolution, recurrent, pooling, normalization, and activation layers. More detailed information about this framework will be provided in the background section.

While Keras2c provides a rich set of tools to deploy AI for several types of microcontrollers, it lacks support for vector code and fixed-point implementation. It uses a few high-level functions incompatible with the Polara runtime environment. Thus, the present paper introduces 'Polara-Keras2c,' a derivative of Keras2c, with improvements on the existing framework to address these limitations.

Vector processing, or vector extensions, enables processors to conduct multiple calculations concurrently, a feature beneficial for data-intensive tasks like scientific simulations and machine learning. This motivated the development of processors supporting vector extensions, such as the RISC-V with Vector Extension, which offers a potent alternative to packed SIMD and GPUs. Unfortunately, at this stage of development, implementations of the RISC-V with vector extensions such as ARA [10] lack full Linux support. Indeed, ARA lacks a memory management unit that is needed to support vector code execution on operating systems like Linux.

Indeed, traditional operating systems utilize virtual memory, providing each process with a unique virtual address space and simplifying application memory access while introducing latency and overheads. Notably, the translation look-aside buffer (TLB) can contribute up to 27% of the energy consumption, while actual computation might use less than 1% for ALU-centric instructions [15]. In bare metal

environments, applications directly access physical memory, bypassing additional translations, improving memory access speed and efficiency, and reducing overheads, thus enhancing execution speed and lowering energy consumption.

Polara-keras2c enhances Keras2c by introducing several key features. First, it supports conversion between fixed-point and floating-point implementation layers, allowing for a flexible approach to model accuracy and computational efficiency - a feature crucial for edge devices with limited computing power. This capability is particularly significant as it enables the framework to adapt to various precision requirements of different AI models, making it uniquely versatile.

Polara-Keras2c also supports fixed-point computations across various layers, such as dense convolutions, activation functions, and other basic operations essential for efficient computing on microcontrollers and other resource-constrained devices. This optimization is significant because it reduces the computational complexity and power consumption, critical factors in edge computing environments. Lastly, the framework allows the generated code to run on bare metal environments seamlessly and supports memory rearrangement and various RISC-V vector kernels. This feature is original and significant, as it ensures that Polara-Keras2c can operate in the most resource-constrained environments while fully leveraging the hardware capabilities of RISC-V-based systems. These enhancements in code generation and library files ensure that Polara-Keras2c achieves its goal of efficiently running fixed-point and vector code on Ara, demonstrating substantial advances over existing frameworks. Such advances are original and beneficial when deploying AI models in edge computing scenarios, where resources are limited, and efficiency is paramount.

Polara-Keras2c is designed to fully harness the RISC-V vector extension, which has not been thoroughly integrated into Linux. By tapping into this feature, our bare metal system offers improved performance and efficiency for AI operations, presenting a more flexible and optimized performance than what a general-purpose OS might provide.

Polara-Keras2c prioritizes swift prototyping, more rapidly accommodating newer instructions and features than conventional OS. This rapidity is vital, especially as our framework is tailored to sync well with Polara.

Our framework offers three main enhancements over traditional machine learning environments like Tensorflow or PyTorch:

- **Bare Metal support**: It uses direct physical memory access to reduce memory overhead and can support a broader range of microcontrollers.
- **RISC-V Vector Optimization**: It leverages the full potential of RISC-V vector extensions for enhanced AI performance.
- **Enhanced Rapid Prototyping**: It supports quickly adopting new features and instructions while integrating well with custom hardware.

The paper unfolds as follows: Section II delves into the backdrop and prior research in the field. The intricacies of the Polara system on a chip (SoC) are unraveled in Section III. Our software blueprint and performance intricacies are elaborated upon in Section IV, followed by an exploration of our testing framework and its outcomes in Section V. The paper culminates with a discussion in Section VI and then a summary and an outlook on prospective endeavors in Section VII.

II. BACKGROUND

Deploying AI functionalities on edge devices, particularly microcontrollers, presents unique challenges, such as limited computational power, constrained memory capacity, and strict energy consumption requirements. Although no universal solution exists for AI execution on microcontrollers [16], several strategies exist to address these challenges. Techniques like quantization and quantization-aware training ensure AI models operate within predefined time and energy constraints. For instance, Hosseini's QS-NAS [17] explores optimal quantization and scaling factors for low-bit-width neural networks, focusing on accuracy and energy efficiency on hardware such as FPGAs.

A. THE NEED TO ACCELERATE AI

There are multiple ways to accelerate AI workload on modern hardware. These often exploit SIMD processing found in array processors, vector processors, and GPUs. SIMD processors are specialized computing units designed to handle multiple data elements concurrently, making them exceptionally apt for vector, matrix, and tensor arithmetic tasks ideal for AI inference. Historically, they have been pivotal in scientific and engineering endeavors that grapple with vast datasets, such as climate simulations, aerospace modeling, and high-resolution image processing [18]. The realm of Artificial Intelligence (AI), with its inherently parallelizable computations, has reinvigorated interest in vector processors.

At the core of many AI operations are neural networks, multi-layered constructs that process input data through a series of mathematical operations. The Fused Multiply Add (FMA) operation is a notable component of deep learning computations. The efficiency of this operation often determines the performance of neural networks, and the Ara processor's capability to handle FMA across extensive vectors of diverse unit sizes (e.g., 8, 16, 32-bit) makes it pertinent for edge AI tasks [19].

The growth in the complexity of AI models, as evidenced by the parameter expansion from GPT to GPT-3 [20], [21] which has grown from 117 million to 175 billion, and the anticipated growth to over one trillion in GPT-4 [22], necessitates computational units capable of handling large-scale computations efficiently with language models like ChatGPT which bodes well with vector acceleration on large servers. Some recent developments in smaller

language models can also be deployed on systems with less computational resources like LLaMA [23].

Moreover, vector processors' energy efficiency is crucial for AI applications on energy-constrained devices. By optimizing computational efficiency and minimizing energy usage, vector processors are pivotal for battery-driven AI implementations.

B. THE RISC-V VECTOR EXTENSION

With its open-source and modular attributes, the RISC-V architecture has quickly emerged as a significant player in modern computing design. A noteworthy feature of this architecture is the RISC-V vector extension [24], which introduces specialized instructions tailored for vector operations. This set of instructions amplifies the potential of RISC-V in executing data-heavy applications like machine learning, image analytics, and computational simulations.

The vector extension enriches the core RISC-V instruction set by enabling vector data processing. Facilitating simultaneous operations on diverse data elements dramatically accelerates computations, especially when handling expansive datasets. A unique flexibility aspect is the support for varying vector lengths, permitting tailored optimization depending on application demands.

In Artificial Intelligence (AI), the RISC-V vector extension markedly elevates processing capabilities, which is particularly beneficial for the parallel processing demands of AI computations. This enhancement is crucial for efficiently performing core neural network functions such as matrix operations, convolutions, and activations, leading to accelerated computational results and significant energy savings. Such improvements are essential for AI systems with stringent resource constraints. The effectiveness of vector processing in boosting both performance and energy efficiency is well-documented in studies like [25], [26], and [27], showcasing the vector extension's substantial impact across various benchmarks.

C. BARE METAL AI RUNTIMES: AN OVERVIEW

Edge devices, particularly microcontrollers, are at the forefront of a technological revolution, enabling AI functionalities closer to data sources. However, deploying such functionalities has unique challenges, historical solutions, and evolving techniques.

AI applications on edge devices and microcontrollers have predominantly utilized ARM Cortex-M series microcontrollers. Notable examples include SAMD51, Appolo3, Spresense, xCORE.ai, and MAX78000 [28]. However, the adoption of RISC-V architecture is rising in this domain. Three primary frameworks are in focus: STM X-Cube-AI, TensorFlow Lite for microcontrollers (TFLite Micro) [29], and keras2c.

Developed by STMicroelectronics, STM X-Cube-AI provides an integrated solution for deploying AI on STM32 microcontrollers. Its distinguishing features include a comprehensive solution incorporating pre-trained models, supportive libraries for leading AI frameworks, and a code generator that translates trained models into deployable C code. It is optimized for performance, specifically tailored for the STM32 microcontroller series, and offers easy integration, compatible with major AI frameworks. Its primary limitation is its exclusivity to STM32 devices. Notable research leveraging X-Cube-AI includes [30], which assessed the tool's efficiency, and [31], which employed it for edge learning on STM32 Nucleo boards.

TFLite Micro [REF] is renowned for its minimal memory footprint, tailored for environments with resource constraints [29]. It is portable and flexible across a range of microcontrollers, and it features optimized kernels designed for efficient computation with support for model quantization. However, it has limited operational support for specific neural network models and it is incompatible with many microcontrollers, especially most RISC-V processors.

Keras2c, a specialized framework, plays a critical role in converting Python-based Keras models into static C code, facilitating the deployment of neural networks in environments where Python is impractical, such as embedded systems and low-resource hardware, a scenario exemplified by Polara's requirements. This framework provides comprehensive support for a variety of layers, including Core Layers (like Dense, Dropout, Activation), Convolutional Layers (Conv1D, Conv2D), and Pooling Layers (MaxPooling, AveragePooling). Additionally, it handles Recurrent Layers (LSTM, GRU), crucial for time-series and sequential data, along with Embedding, Merging, Advanced Activation, Normalization, Noise removal layers, and Wrapper Layers. This extensive support ensures that Keras2c can replicate various functionalities in Keras models. One of the primary benefits of Keras2c is its ability to enable the deployment of complex neural networks in C programming-based environments, especially advantageous in embedded or resource-constrained contexts where the overhead of Python is a limitation. The framework potentially offers improved execution speed and reduced memory usage compared to Python implementations. In terms of operation, Keras2c translates the high-level functionalities of a Keras model into detailed C programming constructs. This intricate process involves mapping each layer and its parameters from the Keras model to equivalent C components, ensuring that the original model's computational flow and data structures are precisely maintained in the C code version.

Other frameworks, such as MicroAI [32], focus on neural network quantization for low-power 32-bit microcontrollers. TinyIREE, another example, is a compiler and runtime designed for bare metal CPUs and microcontrollers. Nonetheless, its compatibility with RISC-V processors remains limited [33].

Various strategies have emerged to enhance ML workloads on CPUs. Some integrate Nvidia's NVDLA accelerator for distinct ML operations in Linux [34], [35]. The max78000 microcontroller is noteworthy for its native AI accelerator,

showcasing substantial performance improvements [36]. Unique methods like blending AI, mathematical models, and firmware for phased array controls are elaborated in [37]. Furthermore, a keyword spotting algorithm on Max78000 highlighted a 40-fold energy efficiency boost using its convolutional neural networks (CNN) accelerator in contrast to its primary processor [38].

The field that focuses on optimization for embedded systems microcontrollers and other embedded systems is known as TinyML. TinyML is enabled by various techniques such as hyperdimensional computing, swapping, attention condensers, constrained neural architecture search, model compression, quantization, once-for-all network, TinyML benchmark, on-device accelerators, and in-processor learning (federated learning) [39].

D. EMBEDDED PLATFORMS FOR AI

Embedded AI platforms are revolutionizing how AI is integrated into compact, energy-efficient devices. This subsection delves into some of the leading platforms in the market, showcasing their distinct features, capabilities, and roles in advancing embedded AI technology.

SAMD51 [40] is a general-purpose Cortex-M4 microcontroller manufactured by Microchip. It has 1MB of flash memory and 256 kB of RAM and can operate at a maximum frequency of 120MHz. This microcontroller is a representative example of a widely used commercially available family of microcontrollers on the market. Motivated by its widespread acceptance and compatibility with CMSIS-NN [41], TensorFlow Lite for Microcontrollers was used to port various neural network models to the selected hardware platforms.

Apollo3 [42] is an ultra-low-power microcontroller developed by Ambiq. Distinctively, it incorporates the proprietary Subthreshold Power Optimized Technology (SPOT) to enhance energy efficiency. Its specifications include 1MB of flash memory and 384 kB of RAM, with a top 96MHz clock frequency. This device is compatible with TensorFlow for Microcontrollers.

Spresense [43] derived from Sony's CXD5602 chip [44] offers a unique architecture. It encompasses a 6-core Cortex-M4, complemented by a single-core Cortex-M0 that operates at a peak frequency of 156MHz. The Cortex-M0 is primarily responsible for system management, utilizing power gating techniques on the Cortex-M4 cores to conserve energy. For neural network computations, 1.5MB SRAM is allocated for weights and activations, whereas another 256 kB system SRAM is present for other tasks. The platform runs on the NuttX real-time operating system (RTOS) and establishes communication in a star topology between its 6 Cortex-M4 cores.

The **PULP (Parallel Ultra Low Power)** processor [45], a joint initiative between the University of Bologna and ETH Zürich, has a commercial counterpart called **GAP8** [46] that was developed by Greenwave Technologies. This 9-core processor boasts a maximum frequency of 250MHz for the fabric controller. Notably, there is a dedicated CNN accelerator within the same interconnect. Neural networks intended for this processor are trained using TensorFlow and processed through the Greenwave Technologies' AutoTiler tool.

xCORE.ai [47], an innovation by XMOS, is designed for high-throughput edge computing. The chip comprises two distinctive tiles endowed with 1MB SRAM, a Vector Processing Unit (VPU), and a 5-stage pipeline that powers eight logical cores. It can reach up to a frequency of 700MHz. The VPU is adept at managing deep tensors, especially those that possess depth multiples of 32. Neural networks for this chip are primarily trained using TensorFlow and later optimized with XMOS's proprietary tools.

MAX78000 [48] presents a dual-core architecture founded on the Cortex-M4 and RISC-V. That platform comprises a robust 64-core CNN accelerator engine that operates at a top frequency of 50MHz as a peripheral to the primary cores. Distinct memory banks are explicitly provisioned for storing neural network weights and biases. Such networks are sculpted on PyTorch, harnessing the custom operations provided by Maxim Integrated.

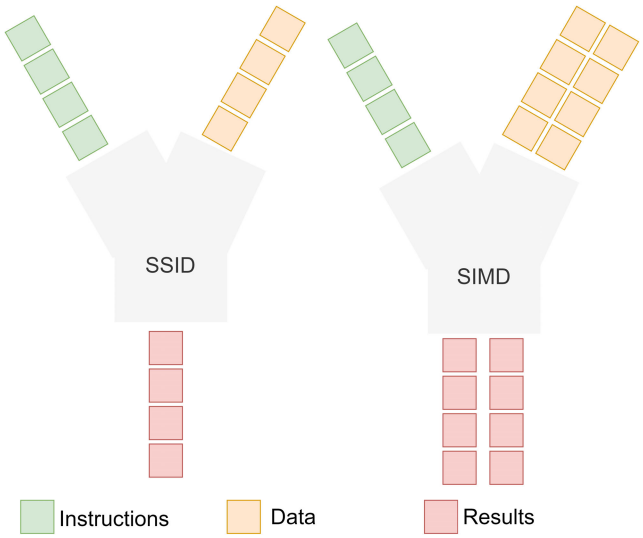


FIGURE 1. Illustration of SIMD operations: processors execute multiple operations simultaneously using identical instructions but on distinct data, leading to a performance enhancement compared to SSID.

III. POLARA

Ara is a distinct RISC-V processor that integrates the CVA6 component, formerly identified as Ariane with a vector engine [49]. The CVA6 is characterized by its 64-bit, 6-stage processing capability with in-order decoding and out-of-order execution. Furthermore, it supports multiply, atomic, and compression extensions. Ara's vector processor is currently designed to function with CVA6 to execute vector codes. Configurations of Ara range from 2 to 16 lanes, supporting vector instructions that can operate on data up to 4096 bits wide. These configurations result in a processor with impressive throughput metrics.

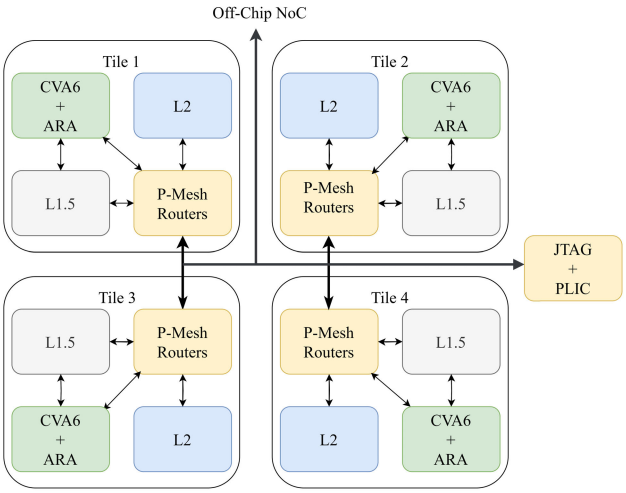


FIGURE 2. System Architecture Overview. The architecture comprises four interconnected tiles, each containing a CVA6 + ARA processing core, L1.5 cache, L2 cache, and P-Mesh routers for inter-tile communication. Tiles are linked to an off-chip Network-on-Chip (NoC) and JTAG + PLIC for debugging and interrupt handling, respectively. The P-Mesh routers manage data traffic within and between tiles, providing scalable, low-latency communication.

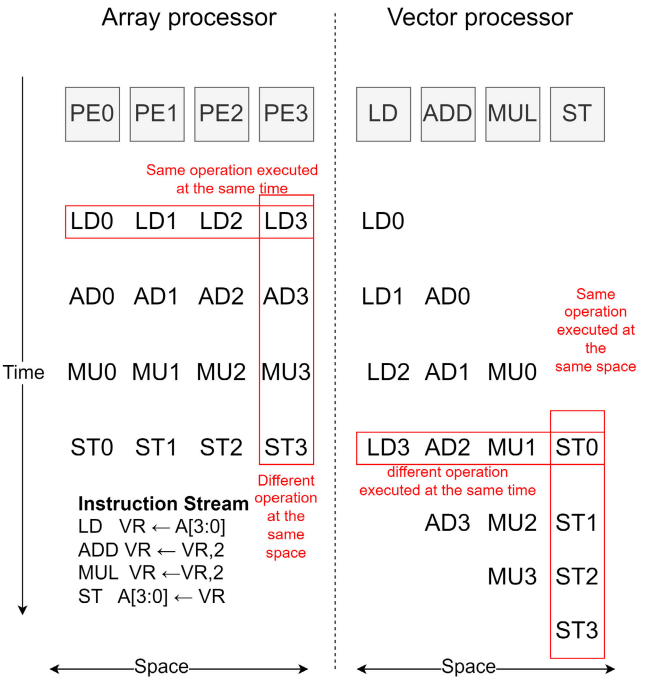


FIGURE 3. Comparison of scalar or array processor with vector processor: the former necessitates full data loading before execution initiation, whereas the latter facilitates execution on the vector in parallel with data loading, reducing execution delays.

Polara as shown in Figure 2, which our work aims to support efficiently, is based on the CVA6-Ara combination. Its efficacy and performance derive from this combination and the SIMD (Single Instruction, Multiple Data) instructions that it offers. Polara integrates a quad-core Ara connected via OpenPiton for cache coherency. In the given architecture, the L1 cache is bifurcated: data (*8 kB*) and instructions (*16 kB*). An intermediate L1.5 cache is *8 kB* per core, followed by an L2 cache of *64 kB*. The Vector Register File (VRF) has a capacity of *2 kB*, subdivided into lanes, each *512 B*.

SIMD represents a parallel processing approach wherein one instruction operates on multiple data points in unison instead of one data element per instruction, thus providing better processing throughput as seen in Figure 1. Its application in AI accelerates data processing, especially for extensive datasets, optimizing algorithmic efficiency. Prominent utilization areas include matrix computational methods and CNNs. With SIMD, concurrent data point processing reduces the time required to compute such operations, from matrix dot products to CNN convolution optimization.

To understand how Polara operates, we must first understand the difference between array and vector processors. Array processors such as x86 architectures with SIMD extensions like Intel's AVX follow a packed-SIMD architecture, where multiple processing elements (PE) operate on data simultaneously, paced by a single control unit, with a fixed vector length determined by the number of PEs. Expanding vector length requires new ISA extensions, with examples including Intel's progression from SSE (128-bit registers) to AVX and AVX-512 (256 and 512-bit registers, respectively), and ARM's Neon (128-bit registers) [10].

Vector processors implement time-multiplexed vector-SIMD instructions, allowing dynamic configuration of vector lengths without the constraints of fixed-size PEs. This architecture is more energy-efficient for long vector operations since it does not require subdividing data into smaller chunks. It can maintain constant control signals throughout the computation, reducing instruction fetch costs [10].

A notable distinction exists between executing traditional CPU vector instructions such as Intel AVX [50] and RISC-V vector instructions [24] and [51]. Figure 3 depicts a typical array processor executing AVX instructions that mandates sequential register loading for every instruction, followed by simultaneous vector element execution. In contrast, the RISC-V requires an initial vector load, which is succeeded by program execution. An advantage of the RISC-V vector is its support for variable-length vector instructions, optimizing resource allocation for datasets of variable dimensions. Utilizing vector instructions instead of fixed SIMD ones, such as AVX, potentially leads to power and area conservation, given the RISC-V vector's reduced requirements for loading compared to AVX [52]. The inherent flexibility of the RISC-V vector also suggests that it might dynamically adapt to available logic units without necessitating code modifications for hardware adjustments.

Polara leverages the OpenPiton framework [53], developed by Princeton University, to implement a multi-core version of the Ara processor. OpenPiton is known for its scalability and architectural versatility, supporting a broad range of core counts from single to many-core configurations. It provides a robust platform for SoC design, offering essential tools and scripts for both ASIC and FPGA implementations.

In Polara, the OpenPiton framework is used to adapt the Ara processor into a scalable multicore architecture. This

adaptation involves integrating multiple instances of the Ara core within a single SoC design, harnessing the scalability offered by OpenPiton. The goal is to achieve a many-core system that can effectively handle the demands of advanced AI and edge computing applications.

The integration with OpenPiton expands core counts and benefits from a comprehensive verification infrastructure and compatibility with mature software tools. This integration enhances the overall capabilities of the Ara-based SoC, particularly in terms of processing power and memory capacity. By employing OpenPiton, Polara aims to deliver a robust, versatile, and scalable platform suitable for various edge computing and AI applications.

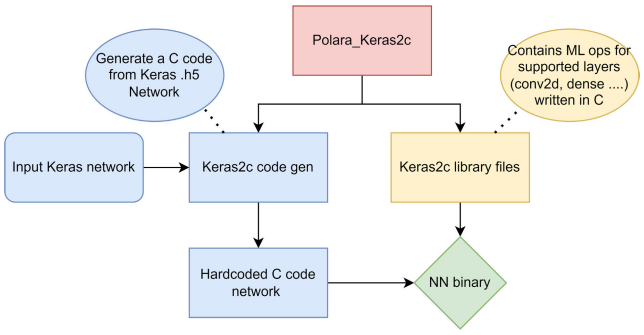


FIGURE 4. Polara-Keras2c system diagram: The code has two parts: a Python code that generates the C code with the layers used, tensor sizes, and library files in C that contain the executable code for supported layers.

IV. SOFTWARE IMPLEMENTATION

To transform a Keras network into a C file, Keras2c requires two positional arguments: 1) $model_path$, which specifies the file path to the saved Keras and.h5 model file, and 2) $function_name$, which denotes the desired name for the resulting C function that outputs the resulting file containing the weight and library calls for the function. Keras2c only supports floating point operation and has no concept of any datatype except floating point.

Similarly, when using Polara-Keras2c, the user must provide $model_path$ and $function_name$ and pass it through to the framework, along with an additional operation flag that denotes the datatype for code generation. The default mode of operation is float, and in that mode, Polara-Keras2c will operate largely as Keras2c but with a few modifications for library function compatible with the Polara bare-metal environment. Polara-Keras2c was modified to allow for the accepting, generating, processing, and executing different datatypes, enabling us to add support for various functionalities such as fixed-point arithmetic and vector arithmetic without retraining a new network. Figure 4 showcases the architecture of Polara-Keras2c. Its source code is available at https://github.com/nizarzarif/polara-keras2c.

Polara-Keras2c has two main parts: a library part and a code generation part. The library part of Polara-Keras2c provides a set of functions and data structures necessary for executing the generated C code on the target microcontroller. This includes functions for loading the model's weights and biases, performing the required operations such as neural network operations (Dense layer, convolutional layers, recurrent layers . . .), activations functions, and core functions (MatMul, additions, matrix multiplications), and outputting the model's predictions. The original Keras2c framework only supports floating point operations. Polara-Keras2c adds support for optimized custom RISC-V vector functions to accelerate ML operations (convolution, MatMul, activations, . . .); it also adds support for fixed-point operations for various layers and functions necessary for end-to-end inference like data type translations, fixed point scaling, overflow, and underflow protection to ensure correct output among many others.

The code generation part is responsible for converting the Keras model into C code that can be executed on the targeted microcontroller. This code includes the model's architecture, weights, and biases and is optimized for efficient execution on resource-constrained microcontrollers. The code generation part also provides options for optimizing the code for size, memory usage. The code generation works by reading all layers of the input keras network. Then, it writes the content of these layers one by one in an output C file while calling the appropriate function from the library file to operate. The original Keras2c only supports floating point operations. Polara-Keras2c adds the options to generate a network using optimized vector features, and fixed-point options, this is done by calling the correct version of the ML operations from the updated library files and transforming the datatype as needed based on the input flag.

These two parts of Polara-Keras2c provide a complete solution for deploying deep-learning models on micro-controllers. By converting Keras models into optimized C code, Polara-Keras2c allows developers to leverage the power of deep learning in resource-constrained environments. Additionally, by providing a library of functions and data structures, Polara-Keras2c simplifies integrating deep learning models into microcontroller-based applications. The architecture of Polara-Keras2c allows for quickly extending support for new functions and new datatypes in both library files and code generations.

A. 2D CONVOLUTION

In deep learning and computer vision, the 2D convolution operation, commonly called Conv2D, stands as a cornerstone. This operation entails applying a compact kernel or filter over an input matrix, such as an image, to yield a feature map. Such an operation excels at discerning local patterns in the input, be it edges or textures. Within CNNs, by stacking multiple Conv2D layers, the network gains the capacity to identify patterns of escalating complexity. The strength of Conv2D is its adeptness at maintaining the spatial correlation between pixels, which renders it exceptionally effective for image recognition such as facial recognition [54] or facial expression recognition [55]. Moreover, by leveraging

parameter sharing, Conv2D layers considerably curtail the model's parameter count, boosting computational prowess and diminishing overfitting risks.

In this research, we present two implementations of Conv2D. The first is a universal rendition, supporting a variety of sizes, strides, and dilations, and that uses the channel-last memory organization (NHWC), termed the scalar implementation. The second version, crafted for optimal performance on RISC-V architectures, is manually written in assembly language compatible with the RISC-V vector extension version 1.0. This vector implementation is optimized explicitly for a stride and dilation of 1 and adheres to the NCHW format–where 'N' stands for the batch size, 'C' for the number of channels (e.g., 3 for RGB, 1 for grayscale), 'H' for the image's height, and 'W' for its width. The development of this vector implementation in handwritten assembly derived from our previous work in [12], rather than through auto-vectorization. This was done to ensure that the code fully capitalizes on the capabilities of the RISC-V vector extension for enhanced computational efficiency.

The Polara-Keras2c framework currently supports only the NHWC memory layout. However, this structure is incompatible with the NCHW Conv2D code, necessitating input and output data transposition to ensure compatibility with all subsequent layers.

The Polara-Keras2c framework supports the NHWC (Channels Last) format while efficiently executing NCHW (Channels First) convolutions when a vector core is available, presenting a unique blend of flexibility and performance optimization. This hybrid approach leverages the inherent advantages of NHWC for scalar CPU operations, such as improved data locality for per-pixel operations and straightforward implementation for algorithms that process images at the pixel level. By accommodating NHWC, the framework ensures compatibility with a wide array of high-level machine learning and image processing APIs, which often default to or support NHWC due to its natural alignment with how images are stored and manipulated in memory. Such a framework significantly simplifies the development and deployment pipeline by eliminating the need for constant data format transformations, streamlining workflows, and reducing the computational overhead associated with format conversion.

On the other hand, for operations that benefit from the parallel processing capabilities of vector cores, such as convolutions, activation, and pooling prevalent in deep neural networks, the framework can temporarily convert data from NHWC to NCHW. This conversion allows it to tap into the computational efficiencies of NCHW on vector-optimized hardware, like RISC-V architectures, where channel data's contiguous storage enables efficient vector register loading and processing. This strategy not only capitalizes on the reduced latency and overhead offered by NCHW for vectorized computations but also maintains the broader applicability and ease of use facilitated by NHWC in scalar processing contexts. By intelligently managing data formats based on the underlying hardware and the nature of the computation, the framework offers a versatile solution that maximizes performance and energy efficiency across a diverse range of computing environments, from high-performance servers to energy-constrained edge devices.

B. TRANSPOSITION

The function `NHWCtoNCHW` shown in Algorithm 1 converts a tensor's memory layout from the NHWC (Height-Width-Channel) format to the NCHW (Channel-Height-Width) format in-place. The function first retrieves the dimensions of the tensor: its height, width, and channel count. It then uses nested loops to iterate over the tensor elements. The function calculates two indices for each element: the current position in the NHWC format and the target position in the NCHW format. Based on these indices, the tensor elements are rearranged to align with the NCHW format. After this transposition, the modified values are transferred back to the original tensor array to complete the transformation. We reuse the same buffer memory for all transpositions to maximize memory efficiency. To do this effectively, we first scan the entire network to determine the largest memory buffer size needed and allocate the required amount. This approach is necessary due to the lack of virtual memory support, which makes dynamic allocation and deletion impossible with the current implementation.

Algorithm 1 In-Situ NHWC to NCHW Conversion

```
procedure NHWCtoNCHW(k2c_tensor* tensor)
    h ← tensor->shape[0]
    w ← tensor->shape[1]
    c ← tensor->shape[2]
    for channel = 0 to c − 1 do
        for height = 0 to h − 1 do
            for width = 0 to w − 1 do
                idx_lhs ← channel × h × w + height × w + width
                idx_rhs ← height × w × c + width × c + channel
                buffer[idx_lhs] ← tensor->array[idx_rhs]
            end for
        end for
    end for
    for i = 0 to h × w × c − 1 do
        tensor->array[i] ← buffer[i]
    end for
end procedure
```

In Algorithm 2, the `TransposeKernel` function transposes the memory layout of convolutional kernels. Typically, these kernels have a format represented by the order: Height (`H`), Width (`W`), Input Channels (`Cin`), and Output Channels (`Cout`). This function aims to adjust the kernel data so the channel dimensions come before the spatial

dimensions. It first retrieves the dimensions of the kernel. The function transfers the data from the kernel to an intermediate buffer using nested loops. During this operation, indices are calculated to guide the data transposition. After reordering the data in the buffer, the adjusted values are copied back to the kernel tensor to finalize the change in memory layout.

Algorithm 2 Kernel Transposition

```
procedure TransposeKernel(k2c_tensor* kernel)
    H, W, Cin, Cout ← kernel->shape
    for h = 0 to H − 1 do
        for w = 0 to W − 1 do
            for cin = 0 to Cin − 1 do
                for cout = 0 to Cout − 1 do
                    idx_lhs ← cout × Cin × H × W + cin × H × W + h × W + w
                    idx_rhs ← h × W × Cin × Cout + w × Cin × Cout + cin × Cout + cout
                    buffer[idx_lhs] ← kernel->array[idx_rhs]
                end for
            end for
        end for
    end for
    for i = 0 to H × W × Cin × Cout − 1 do
        kernel->array[i] ← buffer[i]
    end for
end procedure
```

Finally, in Algorithm 3 `TransposeOutput` function transposes the memory layout of an output tensor from the NCHW format to the NHWC format. In the NCHW format, the tensors sequence is organized by the Number of Channels (Cin), Height, Width, and Output Channels (Cout). The function aims to reorder this sequence to prioritize the spatial dimensions (Height and Width) before the channel dimensions. To achieve this, the function first extracts the tensor's dimensions. It then calculates two indices for each tensor element: `curr_index`, which indicates its current position in the NCHW format, and `target_index`, which denotes its desired position in the NHWC format. Leveraging these indices, tensor values are re-positioned and stored temporarily in a buffer. Once the tensor has been entirely transposed, the rearranged values are copied from the buffer back to the tensor, completing the transposition.

C. FIXED-POINT IMPLEMENTATION

Fixed-point arithmetic, a method for representing and calculating real numbers using integers, offers several advantages in AI and edge computing. This method is particularly beneficial for systems without dedicated floating-point hardware, which is common in microcontrollers and other edge devices. By leveraging fixed-point operations, the Polara-Keras2c framework ensures efficient execution and reduced power consumption, essential in resource-

Algorithm 3 Output Tensor Transposition From NHWC to Custom Format

```
procedure TransposeOutput(k2c_tensor* tensor)
    out_channels ← tensor->shape[3]
    in_channels ← tensor->shape[2]
    out_rows ← tensor->shape[0]
    out_cols ← tensor->shape[1]
                                        ▷ Transposition into buffer
    for k_out = 0 to out_channels − 1 do
        for k_in = 0 to in_channels − 1 do
            for r = 0 to out_rows − 1 do
                for c = 0 to out_cols − 1 do
                    curr_index ← k_out × (in_channels × out_rows × out_cols) + k_in × (out_rows × out_cols) + r × out_cols + c
                    target_index ← r × (out_cols × in_channels × out_channels) + c × (in_channels × out_channels) + k_in × out_channels + k_out
                    buffer[target_index] ← tensor->array[curr_index]
                end for
            end for
        end for
    end for
                                        ▷ Copy transposed data back into tensor
    for i = 0 to out_channels × in_channels × out_rows × out_cols − 1 do
        tensor->array[i] ← buffer[i]
    end for
end procedure
```

constrained environments. This efficiency becomes crucial in edge computing applications, where speed and energy efficiency are paramount [56].

The consistency and determinism of fixed-point arithmetic provide uniform results across different platforms. This aspect is vital in edge computing scenarios, where applications often span diverse devices [57]. Additionally, with their predictable precision, fixed-point numbers are more memory-efficient than floating-point representations, a significant advantage given the limited memory resources of edge devices [58].

Moreover, fixed-point arithmetic in Polara-Keras2c allows for precise control over the allocation of bits to the integer and fractional parts. This control is beneficial for tailoring models to specific application requirements, balancing precision and computational resource usage [59]. The simplicity in hardware implementation due to the absence of special cases like Not a Number (NaN) or infinities and the lack of an exponent and sign bit leads to reduced overhead and more efficient hardware designs [60].

In Algorithm 4, the addition of fixed-point numbers a and b is performed. These numbers are typically represented in a format described by integer bits (m) and fractional bits

Algorithm 4 Fixed-Point Addition

Require: a, b: fixed-point numbers in Qm.n format, m: integer bits, n: fractional bits
Ensure: Result of the fixed-point addition

$qFactor \leftarrow 2^n$
$result \leftarrow (a \ll n) + (b \ll n)$
$maxVal \leftarrow (1 \ll (m + n - 1)) - 1$
$minVal \leftarrow -maxVal - 1$
if $result > maxVal \times qFactor$ **then**
    $result \leftarrow maxVal \times qFactor$ ▷ Handle overflow
else if $result < minVal \times qFactor$ **then**
    $result \leftarrow minVal \times qFactor$ ▷ Handle underflow
end if
$result \leftarrow result \gg n$
return *result*

(n). The function aims to calculate the result of adding two fixed-point numbers while maintaining precision. It first computes the scaling factor $qFactor = 2^n$ to convert the fixed-point numbers to an integer-scaled representation by left-shifting the values of a and b by n bits. The function then adds these scaled values. During this operation, the algorithm checks for potential overflow and underflow conditions, determining if the result exceeds $maxVal = (1 \ll (m + n - 1)) - 1$ or falls below $minVal = -maxVal - 1$. The result is capped or raised to these limits if either condition is met. Finally, the adjusted sum is right-shifted back by n bits to convert it back to the original fixed-point format, finalizing the operation and ensuring the result is within the representable range.

Algorithm 5 Fixed-Point Multiplication

Require: a, b: fixed-point numbers in Qm.n format, m: integer bits, n: fractional bits
Ensure: Result of the fixed-point multiplication

$qFactor \leftarrow 2^n$
$result \leftarrow a \times b$
$result \leftarrow (result \gg n) + ((result \ \& \ (qFactor - 1)) \gg n)$
$maxVal \leftarrow (1 \ll (m + n - 1)) - 1$
$minVal \leftarrow -maxVal - 1$
if $result > maxVal$ **then**
    $result \leftarrow maxVal$ ▷ Handle overflow
else if $result < minVal$ **then**
    $result \leftarrow minVal$ ▷ Handle underflow
end if
return *result*

The framework's adaptability in automatically switching between fixed and floating-point representations as needed provides flexibility, ensuring compatibility with a broader range of models. This flexibility is crucial for running layers that do not yet support fixed-point processing [60].

To enable fixed-point operations, several functions were added, such as fixed-point convolutions, matrix multiplications, bias additions, and activations. However, fixed-point operations are susceptible to overflow and underflow. To prevent these issues, the standard multiplication and addition operations were replaced with safer alternatives, multiplyFixedPoint and addFixedPoint, respectively. Additional functions were also created to convert between floating-point tensors and fixed-point tensors.

Another function *k2c_bias_add_fixed_point* adds bias to a fixed-point convolution output, applying an integer-based computation. Additionally, a fixed-point version of an activation function is invoked to apply non-linear transformations to the fixed-point convolution results.

The code also includes numerous other functions that manage different tensor operations, such as reshaping, transforming data formats, padding, cropping, and upsampling. These operations are essential for constructing neural network architectures and do not inherently specify fixed-point calculations, but they are vital components in preprocessing and postprocessing within neural network layers. This suite of functions is designed to support the development of neural network inference mechanisms on platforms where fixed-point computation is necessary or preferred over floating-point due to hardware constraints. Finally, testing indicates no significant loss of accuracy when using fixed-point arithmetic in simple CNN models on datasets like MNIST. This finding demonstrates that judicious use of fixed-point arithmetic in edge computing may be performed without compromising the model's effectiveness, ensuring that the models are not only efficient but also accurate [61].

V. EXPERIMENTAL RESULTS

In this section, we present a detailed analysis of three experimental approaches and the results obtained, focusing on three aspects: performance scaling of the Polara-Keras2c framework, comparative performance and energy efficiency against other established platforms, and the performance of the fixed-point implementation of Polara-Keras2c.

Firstly, we evaluate the performance scaling of the Polara-Keras2c framework, specifically on Conv2D. This test aims to assess how the framework's performance varies with different input sizes and the number of filters, specifically focusing on the impact of varying input sizes, namely 16×16, 32×32, and 64×64, and the number of filters of 2 to 16. The results are examined to understand the framework's scalability under varying computational loads. This performance scaling is crucial to demonstrate the framework's adaptability and efficiency in handling different neural network architectures and data sizes, as highlighted in the approach to estimate power/energy based on the instruction used in the program [62].

Secondly, we focus on a comparative analysis, where the Polara-Keras2c framework is compared with competitive platforms to gauge its performance and energy efficiency. This comparative test is designed to position the Polara framework within the larger context of existing AI computational platforms. By evaluating the framework's energy

consumption and processing capabilities compared to other market leaders, we aim to provide a comprehensive view of its operational efficiency and potential application scope. This test not only highlights the strengths and limitations of the Polara-Keras2c but also offers valuable insights into the trade-offs between performance and energy efficiency in embedded AI platforms, as task scheduling can have an impact on the speed and efficiency of the program [63].

Lastly, implementing fixed-point arithmetic is critical to deploying deep learning models on edge devices, where computational resources and power efficiency are constrained. In this part of our experiments, we explore the performance of the Polara-Keras2c framework using fixed-point arithmetic, which is pivotal for enhancing the operational efficiency of AI models implemented with limited hardware capabilities.

Through these three tests, we aim to present a holistic evaluation of the Polara-Keras2c framework, underscoring its potential in the rapidly evolving landscape of AI and embedded systems.

A. PERFORMANCE SCALING OF POLARA-KERAS2C

1) EXPERIMENTAL SETUP

Our experimental focus is on the Conv2D layer, which is pivotal in AI applications, especially in computer vision. We gauge performance by the number of clock cycles required for various input sizes and filter numbers. Using Keras, we generated Conv2D layers with random values for weights and biases, varying filter counts (2 to 32), and input sizes (16×16, 32×32, 64×64), all with a kernel size of 3×3. Each layer had a single input channel and default stride and dilation of (1,1).

These layers were processed through Polara-Keras2c to produce C code, which was then compiled using both standard scalar and optimized vector cores for Polara with the LLVM compiler version 14.0.0.1, generating a RISC-V binary. Polara-Keras2c automatically handled data transpositions for the vector setups. We executed the RISC-V binary in Verilator version 4.214 to assess performance, simulating a single-core variant of the Polara environment. This simulation provided the number of clock cycles required to complete each layer and verified the accuracy of the output results. The configuration and toolchain of the simulation can be found at https://github.com/PolyMTL-Gr2m/ara [64].

2) RESULTS

Figure 5 depicts the relationship between clock cycles (in millions) and the number of filters for three different input sizes: 16×16, 32×32, and 64×64.

Figure 6 illustrates the comparative performance of vector and scalar computations across input sizes. The horizontal axis represents the input sizes, ranging from 8×8 to 64×64. The vertical axis represents the computational value, with a scale reaching 1.75×10^7. The plot reveals that vector and scalar computations exhibit relatively close values for smaller input sizes. However, as the input size increases,

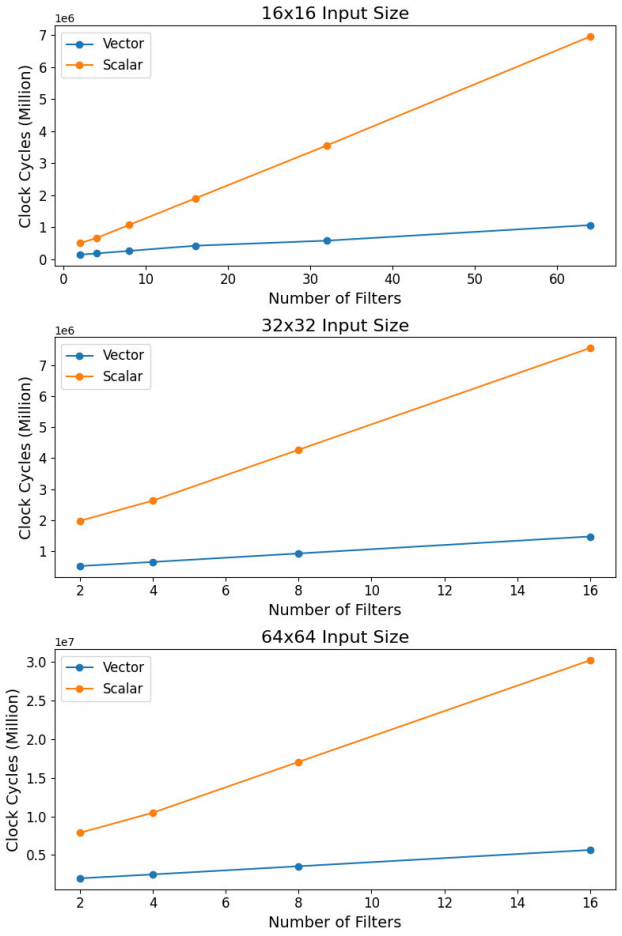


FIGURE 5. Clock cycles in millions versus the number of filters for different input sizes.

scalar computations manifest a steep slope change, notably surpassing the vector computations. Specifically, the scalar value cycle count for an input size of 64×64 is significantly higher than its vector counterpart. This divergence indicates the inefficiencies inherent to larger-scale scalar computations, making vector computations preferable for enhanced performance in such contexts. That is noteworthy, given that the vector implementation needs to transpose the input and kernel before and after the convolution.

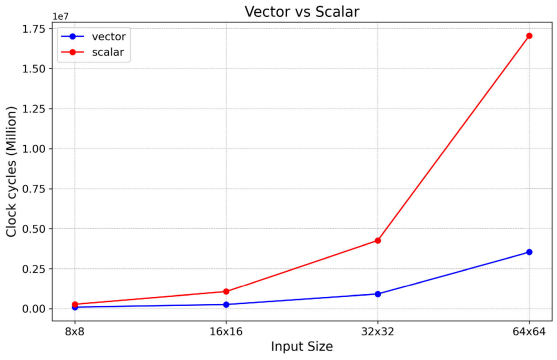


FIGURE 6. Comparison of vector and scalar computational in millions of clock cycles for different input sizes.

B. PERFORMANCE AND ENERGY COMPARISON OF POLARA-KERAS2C WITH COMPETITIVE PLATFORMS

1) EXPERIMENTAL SETUP

The Polara-Keras2c tool is optimized explicitly for the Polara platform and is evaluated against other workflow frameworks to gauge its comparative performance. Nonetheless, given the diverse nature of deployment environments and hardware capabilities, our strategy involves reducing variability by carefully tailoring and optimizing the deployment of the same network model across different tools, each chosen for its compatibility with the respective hardware.

The Polara scalar core and vector core design are based on the Ara core found in [10]. Therefore, we assume that the peak power and scalar power consumption are like those found in the paper which are based on the post-synthesis simulations [10] under typical conditions. By leveraging these detailed and validated measurements, we can provide a reliable estimate of power consumption for the Polara platform.

Performance metrics are assessed using Verilator by the number of clock cycles required to finish one inference. At the same time, measurement is achieved by considering the peak power consumption of the most energy-intensive operation, which, in our case, is the convolutional 2D (conv2d) operations utilizing the vector core. The total estimated energy consumption for one inference, denoted as E, is calculated using the equation:

$$E_{vector} = P_{\text{peak, conv2d}} \times t_{\text{inference}} \tag{1}$$

where $P_{\text{peak, conv2d}}$ represents the peak power consumption observed during conv2d operations, and $t_{\text{inference}}$ is the time required to complete one inference. By focusing on the peak power consumption of the most demanding operation, we estimate conservative energy consumption, encapsulating the upper bound of power usage during the inference process. Unless explicitly stated, we used a vector version of the code in performance and energy testing in the upcoming tests.

A different approach is employed for calculating energy efficiency for scalar operations. The energy consumption, E_{scalar}, is estimated by calculating the power consumption product for scalar operations, P_{scalar}, and the corresponding execution time, t_{scalar}. This is expressed as:

$$E_{\text{scalar}} = P_{\text{scalar}} \times t_{\text{scalar}} \tag{2}$$

2) RESULTS

In our evaluation of different processors, we focused on two main factors: energy usage and performance. To conduct this assessment, we implemented the neural network described in [28] and [65], a face recognition CNN. Using the data provided by the authors, we compared the energy consumption and performance with our software and hardware platform. The model is depicted in Figure 7. Table 1 compares how various AI platforms handle this application, detailing processor type, memory capacity, operating frequency, speed, and efficiency. We estimated the number of inferences completed in one second. We calculated energy efficiency by determining the number of inferences each processor could perform per joule. The results of these calculations are plotted in Figure 8 for processing speed and Figure 9 for energy efficiency.

Figure 8 and Figure 9 show the overall performance and energy of the Polara platform where the higher the number, the better the result. In our comparative analysis, we first consider the computational performance of the processors. Figure 8 quantifies this by detailing the number of inferences per second. The Polara platform result, reported in red, delivers a competitive performance, as its positioning in the graph suggests. It outperforms SAMD51, Appolo3, and Spresense but falls behind the leading trio of the MAX78000, xCORE.ai, and GAP8.

Turning our attention to energy efficiency, Figure 9 provides insights by depicting the number of inferences per joule for each device. Here, the MAX78000 excels, achieving the highest energy efficiency among the evaluated devices. The Polara platform yields moderate results between the highest and lowest performers. Slight advantages are observed for the GAP8 and xCORE.ai processors over the Polara, suggesting their potential suitability for energy-sensitive applications.

C. FIXED-POINT ACCURACY AND PERFORMANCE

To assess the implications of utilizing fixed-point arithmetic in neural network operations, a straightforward CNN model was developed using Keras, a high-level neural networks API, and then deployed using the Polara-Keras2c framework. This deployment aimed to compare the computational performance and precision between models run in standard floating-point precision and those converted to fixed-point arithmetic.

1) EXPERIMENTAL SETUP

Initially, the CNN was trained in a typical floating-point environment using Keras with the entire MNIST dataset [66]. This approach is standard due to its robustness and high accuracy in handling numeric calculations across a wide dynamic range. However, floating-point computations generally demand substantial computational resources, which can be restrictive in resource-constrained environments like embedded systems.

After training and deploying the model in floating-point precision, it was converted to operate in fixed-point format. The transition to fixed-point arithmetic is primarily motivated by the potential gains in computational efficiency. Fixed-point models streamline arithmetic operations by representing numbers at a fixed number of decimal places. This can enhance execution speeds and reduce power consumption, albeit possibly at the cost of introducing quantization errors.

To quantitatively measure the impact of this arithmetic transformation, the performance of the CNN in both operational modes was compared. The focus was placed on the

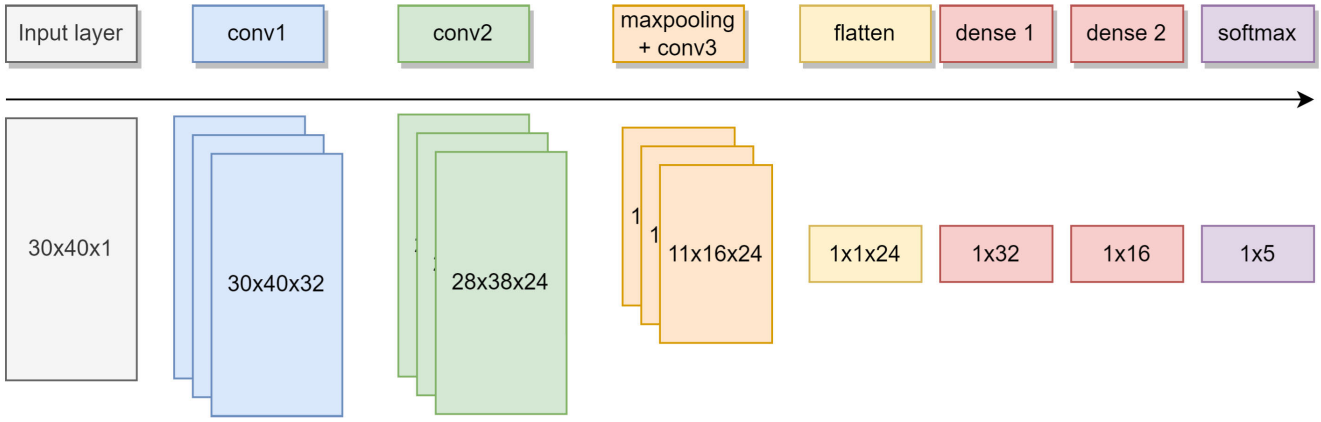


FIGURE 7. Face recognition model used for testing [65].

TABLE 1. Comparison of AI platforms.

Platform	Processor	Memory	Frequency	Special Features
Polara	Quad-core Ara	L1: 8 kB data, 16 kB instructions	750MHz	Vector Register File (VRF), OpenPiton cache coherency
SAMD51 [40]	Cortex-M4	1MB flash, 256 kB RAM	120MHz	CMSIS-NN [66], TensorFlow Lite compatibility
Apollo3 [42]	Ambiq Apollo3 Blue ARM Cortex-M4	1MB flash, 384 kB RAM	96MHz	SPOT technology for enhanced energy efficiency
Spresense [43]	6-core Cortex-M4	1.5MB SRAM, 256 kB system SRAM	156MHz	NuttX RTOS, star topology communication
PULP/GAP8 [46]	9-core	-	250MHz	Dedicated CNN accelerator, TensorFlow processing
xCORE.ai [47]	16-core (2 tiles of 8 logical cores each)	2MB SRAM (1MB per tile)	700MHz	VPU for deep tensors, TensorFlow optimization, 5-stage pipeline
MAX78000 [48]	Cortex-M4 and RISC-V	-	50MHz	64-core CNN accelerator engine

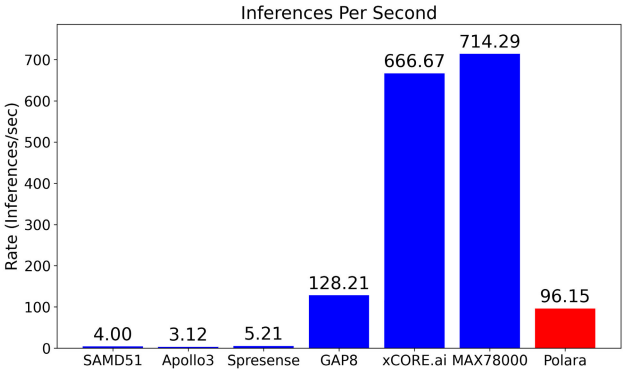


FIGURE 8. Number of inferences per second.

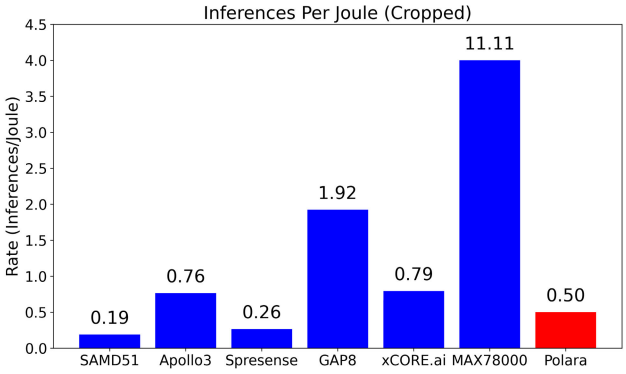


FIGURE 9. Number of inferences per Joule.

accuracy of the model outputs and the execution speed, crucial metrics that determine the practicality of deploying fixed-point arithmetic in operational environments. This analysis is vital for evaluating the trade-offs inherent in fixed-point computation, mainly how it affects the overall effectiveness of the neural network in real-world applications.

Figure 10 depicts a convolutional neural network (CNN) used primarily for image classification. Below is a short description of each layer within the network:

Input Layer: Accepts a 28 × 28 pixel image and serves as the entry point for data processing.

Convolutional Layer (Conv2D): Directly beneath the input layer, it employs 16 filters of 3 × 3 to perform feature extraction using the ReLU activation function, enhancing the network's ability to learn complex patterns.

Pooling Layer (MaxPooling): Reduces the spatial dimensions of the data using 2 × 2 max pooling, which simplifies the output and helps in reducing overfitting.

Flatten Layer: Converts the pooled output into a one-dimensional vector, enabling the transition to fully connected layers.

First Dense Layer: Processes the flattened data through 64 neurons with ReLU activation to further refine features.

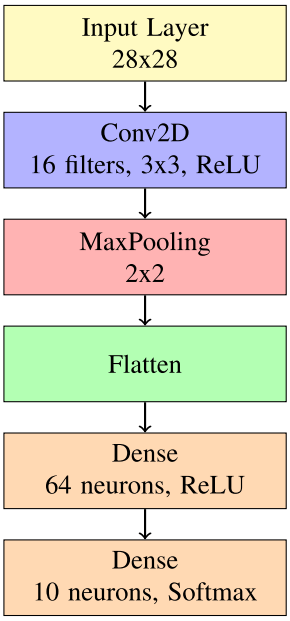


FIGURE 10. A schematic diagram of the CNN model layers, including the input layer.

Output Layer (Second Dense Layer): Comprises 10 neurons, each representing a class, and utilizes the Softmax activation to output a probability distribution over the classes.

Arrows between the layers illustrate the direction of data flow, emphasizing the sequential processing from input to output, where each layer progressively abstracts and refines information, culminating in classification.

2) RESULTS

The effectiveness of fixed-point arithmetic in neural network operations was evaluated using a model -shown in Figure 10- trained on the MNIST dataset. This dataset, which includes handwritten digits, is widely used for benchmarking classification algorithms in machine learning. Initially, the model was trained in a standard floating-point environment using Keras, achieving an accuracy of 99.7%. Following the training, the model was converted to operate in a fixed-point format using Polara-keras2c. This conversion was implemented to explore potential gains in computational efficiency that fixed-point arithmetic offers, especially on platforms with limited hardware capabilities that do not natively support floating-point operations.

The transition from floating-point to fixed-point involved quantizing the model's parameters, such as weights and biases, from floating-point representations to integer representations with a fixed number of decimal places. This process was carefully managed to minimize the loss of precision, ensuring that the quantization did not significantly impact the model's ability to predict outcomes accurately.

Upon converting the model to fixed-point arithmetic, it was tested again using the same MNIST dataset to evaluate any changes in performance. The fixed-point model achieved an accuracy of 99.67%, only a slight decrease from the original floating-point model's accuracy. Regarding performance, running the same network in fixed-point needed 27.7 million cycles compared to 7.6 million cycles in the floating implementation due to safe multiplication and accumulation that prevents underflow or overflow.

VI. DISCUSSION

Summarizing our experimental exploration of the Polara-Keras2c framework, we uncovered significant insights into its performance scaling and energy efficiency. For example, when processing 64×64 input sizes convolutions, Polara-Keras2c demonstrated performance gains of up to 4.81 times faster compared to scalar processing as seen in Figure 6. The vector approach notably excels in managing larger input sizes and a more significant number of filters, showcasing the framework's robust scalability.

In all tested scenarios, the vector approach consistently outshined scalar systems, necessitating fewer clock cycles for operation. This efficiency gap widens with an increasing count of filters, underscoring the vector approach's superior performance in various situations. As illustrated in Figure 5, there is an apparent linear progression in performance scaling corresponding to the increase in the number of filters.

The Polara platform's edge lies in its utilization of a vector processor, providing a flexible computing approach in contrast to hardware-specific solutions like Max78000, xCore.ai, and GAP8. While these platforms lead in energy efficiency and overall performance, they are somewhat limited by their reliance on fixed-function blocks. In contrast, Polara's vector processor supports dynamic optimization for various tasks such as in high-performance computing [67], image processing [68], highly-parallel workloads and anywhere traditional vector extension like AVX or arm SVE2 can be used [49], essentially anywhere SIMD can be used, and not limited to specific type of neural networks. This adaptability is crucial in situations that demand variable workload characteristics or require algorithm updates post-deployment. Moreover, unlike platforms that rely primarily on quantized neural networks, Polara supports both floating-point precision and quantized models, showcasing the framework's broad compatibility across computational environments. This adaptability is essential for applications where precision needs may vary, or where models require post-deployment updates or modifications.

The effectiveness of Polara's fixed-point implementation is further underscored by a minimal accuracy reduction of only 0.03% on the MNIST dataset. This negligible impact suggests that fixed-point arithmetic can be a practical and efficient alternative to floating-point computations in certain edge computing applications, where computational efficiency and energy savings are prioritized.

Furthermore, these findings highlight the potential of fixed-point arithmetic to enable the broader adoption of advanced neural network models on edge devices. It opens

up avenues for further research into optimizing quantization strategies to minimize accuracy loss while maximizing computational efficiency. This balance is crucial for expanding machine learning capabilities in embedded systems, paving the way for more sophisticated AI applications that are both power-efficient and performant.

Polara-Keras2c exhibits a remarkable compatibility spectrum, extending from scalar processors to vector-accelerated cores and traditional computing systems. This wide-ranging applicability, encompassing both bare metal and OS-controlled applications, firmly establishes its place in the dynamic realm of AI and embedded systems. The modification added in Polara-keras2c allows us to quickly expand the capability of the framework with additional support for different data types, hardware optimized version of various deep learning operations for specific hardware and adding preprocessing and postprocessing step for every layer if needed, allowing for rapid prototyping.

The integration of RISC-V vector extension support and optimization for the Polara architecture in Polara-Keras2c not only addresses the technical challenges of edge computing but also opens up new possibilities for AI applications in this domain. By significantly reducing execution time and power consumption, Polara-Keras2c enables the deployment of more complex AI models on edge devices, fostering innovation in real-time processing and decision-making applications.

VII. CONCLUSION AND FUTURE WORK

This paper introduced "Polara-Keras2c," a framework designed to deploy AI models on edge devices efficiently. Central to its innovation is the ability to translate Keras deep learning models into optimized C code, facilitating operation in resource-constrained environments. The framework leverages the capabilities of RISC-V vector extensions, demonstrating significant enhancements in processing power, which is critical for applications in edge computing. Comparative evaluations have highlighted the framework's commendable performance and energy efficiency, making it a noteworthy contribution to the field.

Looking forward, there are several potential avenues for improvement in Polara-Keras2c. Expanding support for a more comprehensive array of AI models and datasets could enhance the framework's versatility, catering to a broader spectrum of edge computing applications. Adding quantization support allows fewer bits to represent information, shrinks model size, and efficient deployment on devices with limited memory. This also translates to faster processing due to simpler calculations, boosting real-time performance. Further, adapting the framework for a diverse range of hardware platforms and processors could increase its applicability, addressing the varied needs of edge computing environments. Energy efficiency remains a crucial area for improvement, especially for applications in battery-powered and remote devices where power conservation is paramount.

In essence, Polara-Keras2c is a promising solution for the challenges of AI processing in edge computing. It offers a solid foundation with multiple possibilities for enhancements, reflecting the dynamic nature of research and development in edge AI.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[2] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C Van Esesn, A. A S. Awwal, and V. K. Asari, "The history began from AlexNet: A comprehensive survey on deep learning approaches," 2018, *arXiv:1803.01164*.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.

[4] D. P Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[6] Z. Chang, S. Liu, X. Xiong, Z. Cai, and G. Tu, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13849–13875, Sep. 2021.

[7] I. L. Orăşan, C. Seiculescu, and C. D. Căleanu, "A brief review of deep neural network implementations for ARM Cortex-M processor," *Electronics*, vol. 11, no. 16, p. 2545, Aug. 2022.

[8] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do imagenet classifiers generalize to imagenet?" in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5389–5400.

[9] OpenHW Group. (2023). *CORE-V Polara APU*. Accessed: Feb. 19, 2024. [Online]. Available: https://github.com/openhwgroup/core-v-polara-apu

[10] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, "Ara: A 1-GHz+ scalable and energy-efficient RISC-V vector processor with multiprecision floating-point support in 22-nm FD-SOI," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 530–543, Feb. 2020.

[11] M. AskariHemmat, T. Dupuis, Y. Fournier, N. El Zarif, M. Cavalcante, M. Perotti, F. Gürkaynak, L. Benini, F. Leduc-Primeau, Y. Savaria, and J.-P. David, "Quark: An integer RISC-V vector processor for sub-byte quantized DNN inference," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2023, pp. 1–5.

[12] T. Dupuis, Y. Fournier, M. AskariHemmat, N. El Zarif, F. Leduc-Primeau, J. P. David, and Y. Savaria, "Sparq: A custom RISC-V vector processor for efficient sub-byte quantized inference," in *Proc. 21st IEEE Interregional NEWCAS Conf. (NEWCAS)*, 2023, pp. 1–5.

[13] F. Tonini, C. Natalino, D. A. Temesgene, Z. Ghebretensaé, L. Wosinska, and P. Monti, "Benefits of pod dimensioning with best-effort resources in bare metal cloud native deployments," *IEEE Netw. Lett.*, vol. 5, no. 1, pp. 41–45, Mar. 2023.

[14] R. Conlin, K. Erickson, J. Abbate, and E. Kolemen, "Keras2c: A library for converting Keras neural networks to real-time compatible C," *Eng. Appl. Artif. Intell.*, vol. 100, Apr. 2021, Art. no. 104182.

[15] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 11, pp. 2629–2640, Nov. 2019.

[16] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, "Artificial intelligence in the IoT era: A review of edge ai hardware and software," in *Proc. 31st Conf. Open Innov. Assoc. (FRUCT)*, 2022, pp. 320–331.

[17] M. Hosseini and T. Mohsenin, "QS-NAS: Optimally quantized scaled architecture search to enable efficient on-device micro-AI," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 4, pp. 597–610, Dec. 2021.

[18] J. Mielikainen, B. Huang, H. A. Huang, and M. D. Goldberg, "Improved GPU/CUDA based parallel weather and research forecast (WRF) single moment 5-class (WSM5) cloud microphysics," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 4, pp. 1256–1265, Aug. 2012.

[19] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "AI accelerator survey and trends," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, 2021, pp. 1–9.

[20] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI, San Francisco, CA, USA, 2018. [Online]. Available: https://www.openai.com/research

[21] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, and A. Askell, "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901.

[22] OpenAI et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.

[23] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[24] OpenHW Group. (2024). *RISC-V Vector Specification*. Accessed: Jun. 6, 2024. [Online]. Available: https://github.com/riscv/riscv-v-spec/tree/v1.0

[25] I. Al Assir, M. El Iskandarani, H. R. Al Sandid, and M. A. R. Saghir, "Arrow: A RISC-V vector accelerator for machine learning inference," 2021, *arXiv:2107.07169*.

[26] P. Vizcaino, G. Ieronymakis, N. Dimou, V. Papaefstathiou, J. Labarta, and F. Mantovani, "Short reasons for long vectors in HPC CPUs: A study based on RISC-V," in *Proc. Workshops Int. Conf. High Perform. Comput., Netw., Storage, Anal.*, 2023, pp. 1543–1549.

[27] J. K. L. Lee, M. Jamieson, N. Brown, and R. Jesus, "Test-driving RISC-V vector hardware for HPC," 2023, *arXiv:2304.10319*.

[28] M. Giordano, L. Piccinelli, and M. Magno, "Survey and comparison of milliwatts micro controllers for tiny machine learning at the edge," in *Proc. IEEE 4th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2022, pp. 94–97.

[29] *Tensorflow Lite for Microcontrollers*. Accessed: Jun. 6, 2024. [Online]. Available: https://github.com/tensorflow/tflite-micro

[30] V. Falbo, T. Apicella, D. Aurioso, L. Danese, F. Bellotti, R. Berta, and A. De Gloria, "Analyzing machine learning on mainstream microcontrollers," in *Applications in Electronics Pervading Industry, Environment and Society*. Berlin, Germany: Springer, 2020, pp. 103–108.

[31] F. Sakr, F. Bellotti, R. Berta, and A. De Gloria, "Machine learning on mainstream microcontrollers," *Sensors*, vol. 20, no. 9, p. 2638, May 2020.

[32] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and deployment of deep neural networks on microcontrollers," *Sensors*, vol. 21, no. 9, p. 2984, Apr. 2021.

[33] H.-I. Cindy Liu, M. Brehler, M. Ravishankar, N. Vasilache, B. Vanik, and S. Laurenzo, "TinyIREE: An ML execution environment for embedded systems from compilation to deployment," 2022, *arXiv:2205.14479*.

[34] D. Giri, K.-L. Chiu, G. Eichler, P. Mantovani, and L. P. Carloni, "Accelerator integration for open-source SoC design," *IEEE Micro*, vol. 41, no. 4, pp. 8–14, Jul. 2021.

[35] S.-C. Luo, K.-C. Chang, P.-W. Chen, and Z.-H. Chen, "Configurable deep learning accelerator with bitwise-accurate training and verification," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2022, pp. 1–4.

[36] A. Moss, H. Lee, L. Xun, C. Min, F. Kawsar, and A. Montanari, "Ultra-low power DNN accelerators for IoT: Resource characterization of the MAX78000," in *Proc. 20th ACM Conf. Embedded Networked Sensor Syst.*, 2022, pp. 934–940.

[37] R. Colella, L. Spedicato, L. Laqintana, and L. Catarinucci, "Inertially controlled two-dimensional phased arrays by exploiting artificial neural networks and ultra-low-power AI-based microcontrollers," *IEEE Access*, vol. 11, pp. 23474–23484, 2023.

[38] M. G. Ulkar and O. E. Okman, "Ultra-low power keyword spotting at the edge," 2021, *arXiv:2111.04988*.

[39] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 4, pp. 1595–1623, Apr. 2022.

[40] Mouser Electronics. (2019). *SAM D5x/E5x Family Data Sheet*. Accessed: Feb. 15, 2023. [Online]. Available: https://www.mouser.com/datasheet/2/268/60001507A-1130176.pdf

[41] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm Cortex-M CPUs," 2018, *arXiv:1801.06601*.

[42] Ambiq Micro. (2022). *Apollo3 Blue MCU Data Sheet*. Accessed: Feb. 15, 2023. [Online]. Available: https://ambiq.com/wp-content/uploads/2020/08/Apollo3-Blue-SoC-Datasheet.pdf

[43] Adafruit Industries. *Product 4419 Datasheet*. Accessed: Feb. 15, 2023. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/4419_Web.pdf

[44] Sony Semiconductor Solutions. *CXD5602 User Manual*. Accessed: Feb. 15, 2023. [Online]. Available: https://www.sony-semicon.com/files/62/pdf/p-28_CXD5602_user_manual.pdf

[45] D. Rossi, F. Conti, M. Eggiman, A. D. Mauro, G. Tagliavini, S. Mach, M. Guermandi, A. Pullini, I. Loi, J. Chen, E. Flamand, and L. Benini, "Vega: A ten-core SoC for IoT endnodes with DNN acceleration and cognitive wake-up from MRAM-based state-retentive sleep mode," *IEEE J. Solid-State Circuits*, vol. 57, no. 1, pp. 127–139, Jan. 2022.

[46] GreenWaves Technologies. *GAP8 Datasheet*. Accessed: Feb. 2, 2023. [Online]. Available: https://gwt-website-files.s3.amazonaws.com/gap8_datasheet.pdf

[47] Pawpaw Technology. *XU316-1024-QF60B-PP24 Datasheet*. Accessed: Feb. 15, 2023. [Online]. Available: https://www.pawpaw.cn/media/documents/2022-06/XU316-1024-QF60B-PP24_Datasheet.pdf

[48] Analog Devices. (2021). *MAX78000 Data Sheet*. Accessed: Feb. 15, 2023. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX78000.pdf

[49] M. Perotti, M. Cavalcante, N. Wistoff, R. Andri, L. Cavigelli, and L. Benini, "A 'new ara' for vector computing: An open source highly efficient RISC-V V 1.0 vector processor design," in *Proc. IEEE 33rd Int. Conf. Application-Specific Syst., Architectures Processors (ASAP)*, Jul. 2022, pp. 43–51.

[50] Intel Corporation. (2023). *Intel 64 and IA-32 Architectures Software Developer's Manual*. Combined Volumes:1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/671200

[51] E. Cui, T. Li, and Q. Wei, "RISC-V instruction set architecture extensions: A survey," *IEEE Access*, vol. 11, pp. 24696–24711, 2023.

[52] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller, "Characterizing the energy consumption of data transfers and arithmetic operations on x86–64 processors," in *Proc. Int. Conf. Green Comput.*, Aug. 2010, pp. 123–133.

[53] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, S. Payne, and D. Wentzlaff, "Openpiton: An open source manycore research framework," in *Proc. 21st Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, 2016, pp. 1–16.

[54] M. Coskun, A. Uçar, Ö. Yildirim, and Y. Demir, "Face recognition based on convolutional neural network," in *Proc. Int. Conf. Modern Electr. Energy Syst. (MEES)*, Nov. 2017, pp. 376–379.

[55] N. E. Zarif, L. Montazeri, F. Leduc-Primeau, and M. Sawan, "Mobile-optimized facial expression recognition techniques," *IEEE Access*, vol. 9, pp. 101172–101185, 2021.

[56] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[57] S. Mittal, "A survey on optimized implementation of deep learning models on the NVIDIA Jetson platform," *J. Syst. Archit.*, vol. 97, pp. 428–442, Aug. 2019.

[58] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[59] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Seoul, South Korea, 2019, pp. 3009–3018, doi: 10.1109/ICCVW.2019.00363.

[60] J. Lin, W. Chen, Y. Lin, and J. Cao, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 2849–2858.

[61] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[62] M. McKeown, A. Lavrov, M. Shahrad, P. J. Jackson, Y. Fu, J. Balkind, T. M. Nguyen, K. Lim, Y. Zhou, and D. Wentzlaff, "Power and energy characterization of an open source 25-core manycore processor," in *Proc. HPCA*, 2018, pp. 762–775.

[63] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-throughput CNN inference on embedded arm big.little multicore processorss," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2254–2267, Oct. 2020.

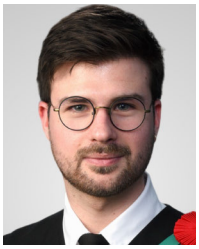
[64] PolyMTL-GR2M. (2024). *ARA: PolyMTL-GR2M*. Accessed: Jul. 8, 2024. [Online]. Available: https://github.com/PolyMTL-Gr2m/ara

[65] M. Giordano, P. Mayer, and M. Magno, "A battery-free long-range wireless smart camera for face detection," in *Proc. 8th Int. Workshop Energy Harvesting Energy-Neutral Sens. Syst.*, Nov. 2020, pp. 29–35.

[66] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.
[67] F. Minervini, O. Palomar, O. Unsal, E. Reggiani, J. Quiroga, J. Marimon, C. Rojas, R. Figueras, A. Ruiz, A. Gonzalez, and J. Mendoza, "Vitruvius+: An area-efficient RISC-V decoupled vector coprocessor for high performance computing applications," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 2, pp. 1–25, Jun. 2023.
[68] R.-S. Li, P. Peng, Z.-Y. Shao, H. Jin, and R. Zheng, "Evaluating RISC-V vector instruction set architecture extension with computer vision workloads," *J. Comput. Sci. Technol.*, vol. 38, no. 4, pp. 807–820, Jul. 2023.

NIZAR EL ZARIF received the bachelor's degree in electronics and communications engineering from Beirut Arab University, Lebanon, in 2011, the master's degree in electrical and computer engineering from American University of Beirut, Lebanon, in 2015, and the Ph.D. degree from the Polystim Neurotech Laboratory, Polytechnique Montréal, Canada, in 2022. He is currently a Postdoctoral Fellow with Polytechnique Montréal. His research interests include high-performance computing, machine learning, embedded system design, computer vision, and parallel computing.

MOHAMMADHOSSEIN ASKARI HEMMAT received the Ph.D. degree from Polytechnique Montréal, Canada, in 2023. He was involved in designing hardware and algorithm for efficient deep neural networks with Polytechnique Montréal. He is currently an AI Research Scientist with Deeplite. His research interests include designing efficient deep neural networks, computer architecture, and machine learning.



THEO DUPUIS received the University Technology Diploma degree in electrical engineering and industrial computer science from the University de Tours, in 2019, the Engineering degree (M.Sc.) in electrical engineering from INSA Lyon, France, in 2022, and the master's degree in electrical engineering, specializing in digital electronics, low-level computing, processor architecture, and machine learning from Polytechnique Montréal, Canada, in 2023. He engaged in a research project on mixed hardware/software acceleration of sub-byte operations on a RISC-V vector processor for quantized neural networks applications, showcasing skills in hardware/software acceleration.

JEAN-PIERRE DAVID received the Ph.D. degree from the Université Catholique de Louvain, Louvain-la-Neuve, Belgium, in 2002. Following that, he was an Assistant Professor with the Université de Montréal, Montreal, QC, Canada, for three years before transitioning to Polytechnique Montréal, Montreal, in 2006. In 2021, he attained the position of a Full Professor. His research interests include digital system design, reconfigurable computing, high-level synthesis, dedicated arithmetic, and practical applications, including signal processing, real-time simulation, and network communications. Since 2014, he has delved into low-precision arithmetic for neural networks. Notably, he authored "Binaryconnect: Training Deep Neural Networks with Binary Weights During Propagations," a seminal contribution to the field of binary neural networks.

YVON SAVARIA (Life Fellow, IEEE) received the B.-Ing. and M.Sc. degrees in electrical engineering from Polytechnique Montréal, in 1980 and 1982, respectively, and the Ph.D. degree in electrical engineering from McGill University, in 1985.

Since 1985, he has been with Polytechnique Montréal, where he is currently a Professor with the Department of Electrical Engineering. Since June 2019, he has been the NSERC-Kaloom-Intel-Noviflow (KIN) Chair Professor. He is the Co-Director of the Regroupement Strategique en Microelectronique du Quebec (RESMIQ) and a member of the Ordre des Ingenieurs du Quebec (OIQ). He has been a consultant or was sponsored for carrying out research with Bombardier, Buspass, CNRC, Design Workshop, Dolphin, DREO, Ericsson, Genesis, Gennum, Huawei, Hyperchip, Intel, ISR, Kaloom, LTRIM, Miranda, MiroTech, Nortel, Octasic, PMC-Sierra, Space Codesign, Technocap, Thales, Tundra, and Wavelite. He has carried out work in several areas related to microelectronic circuits and microsystems, such as testing, verification, validation, clocking methods, defect and fault tolerance, effects of radiation on electronics, high-speed interconnects, and circuit design techniques, CAD methods, reconfigurable computing and applications of microelectronics to telecommunications, aerospace, image processing, video processing, radar signal processing, and the acceleration of digital signal processing. He holds 16 patents, published 211 journal articles and 495 conference papers, and the thesis advisor of 190 graduate students who completed their studies. He is also involved in several projects related to embedded systems in aircraft, wireless sensor networks, virtual networks, software-defined networks, machine learning (ML), embedded ML, computational efficiency, and application-specific architecture design. He is a fellow of Canadian Academy of Engineering. In 2001, he was awarded the Tier 1 Canada Research Chair on the design and architecture of advanced microelectronic systems which he held until June 2015. He also received the Synergy Award from the Natural Sciences and Engineering Research Council of Canada in 2006. He was the Program Co-Chair of NEWCAS'2018 and the General Chair of NEWCAS'2020.