IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 April 2024, accepted 15 May 2024, date of publication 23 May 2024, date of current version 3 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3404878

RESEARCH ARTICLE

Flexible and Fully Quantized Lightweight TinyissimoYOLO for Ultra-Low-Power Edge Systems

JULIAN MOOSMANN[1], (Graduate Student Member, IEEE), HANNA MÜLLER[1], (Graduate Student Member, IEEE), NICKY ZIMMERMAN[2], (Graduate Student Member, IEEE), GEORG RUTISHAUSER[1], (Graduate Student Member, IEEE), LUCA BENINI[1,3], (Fellow, IEEE), AND MICHELE MAGNO[1], (Senior Member, IEEE)

[1]Department of Information Technology and Electrical Engineering, ETH Zürich, 8092 Zürich, Switzerland
[2]IDSIA, Università della Svizzera Italiana, 6900 Lugano, Switzerland
[3]Department of Electrical, Electronic and Information Engineering, University of Bologna, 40126 Bologna, Italy

Corresponding author: Julian Moosmann (julian.moosmann@pbl.ee.ethz.ch)

This work was supported by armasuisse Science and Technology.

ABSTRACT This paper deploys and explores variants of TinyissimoYOLO, a highly flexible and fully quantized ultra-lightweight object detection network designed for edge systems with a power envelope of a few milliwatts. With experimental measurements, we present a comprehensive characterization of the network's detection performance, exploring the impact of various parameters, including input resolution, number of object classes, and hidden layer adjustments. We deploy variants of TinyissimoYOLO on state-of-the-art ultra-low-power extreme edge platforms, presenting a detailed comparison on latency, energy efficiency, and their ability to efficiently parallelize the workload. In particular, the paper presents a comparison between a RISC-V-based parallel processor (GAP9 from GreenWaves Technologies) with and without use of its on-chip hardware accelerator, an ARM Cortex-M7 core (STM32H7 from ST Microelectronics), two ARM Cortex-M4 cores (STM32L4 from ST Microelectronics and Apollo4b from Ambiq), and a multi-core platform aimed at edge AI applications with a CNN hardware accelerator (MAX78000 from Analog Devices). Experimental results show that the GAP9's hardware accelerator achieves the lowest inference latency and energy at 2.12 ms and 150 µJ respectively, which is around 2x faster and 20% more energy efficient than the next best platform, the MAX78000. The hardware accelerator of GAP9 can even run an increased resolution version of TinyissimoYOLO with 112 × 112 pixels and 10 detection classes within 3.2 ms, consuming 245 µJ. We also deployed and profiled a multi-core implementation on GAP9 at different core voltages and frequencies, achieving 11.3 ms with the lowest-latency and 490 µJ with the most energy-efficient configuration. With this paper, we demonstrate the flexibility of TinyissimoYOLO and prove its detection accuracy on a widely used detection dataset. Furthermore, we demonstrate its suitability for real-time ultra-low-power edge inference.

INDEX TERMS YOLO, ML, computer vision, object detection, hardware accelerator, microcontroller, quantization, quantization-aware training, network deployment, network deployment evaluation.

I. INTRODUCTION

With the widespread adoption of edge devices, particularly microcontroller unit (MCU) based nodes, the Internet of Things (IoT) is revolutionizing various domains including

The associate editor coordinating the review of this manuscript and approving it for publication was Md. Arafatur Rahman.

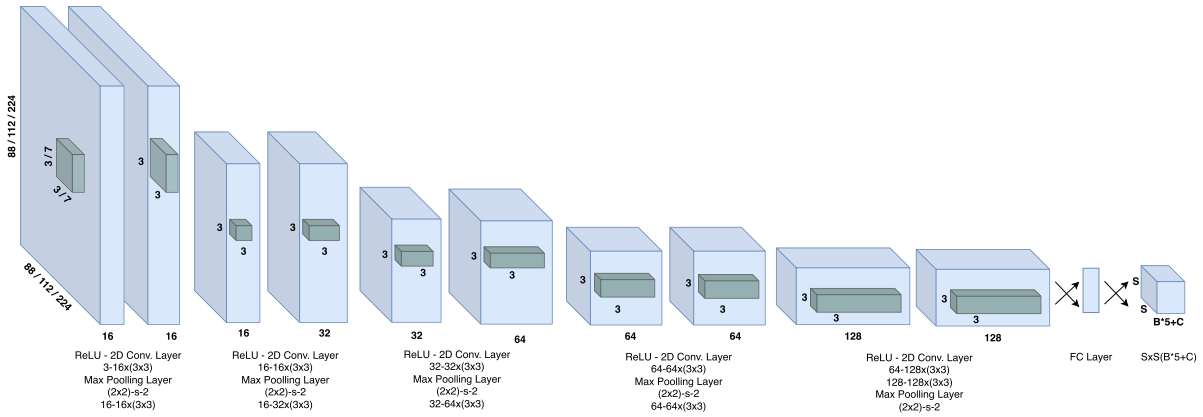

FIGURE 1. TinyissimoYOLO as deployed on GAP9 (both the 9-core cluster and the hardware accelerator) by this paper.

health monitoring [1], [2], smart manufacturing [3], and home automation [4]. These low-power devices enable increased automation, cost reduction, bandwidth optimization, and enhanced privacy by processing collected data *on the edge*, i.e., directly on the sensor node. However, the limited memory and computing resources of MCUs pose a significant challenge when it comes to deploying advanced machine learning models. Overcoming this challenge and enabling efficient machine learning on resource-constrained devices is a crucial area of research in embedded wireless sensor systems, as illustrated by [5].

A foundational element of many IoT applications is the extraction of semantic information about the environment with image sensors. Specifically, object detection [6], [7], [8], the vital task of identifying and precisely localizing objects within a given image, plays a pivotal role in a wide range of systems. For instance, in the realm of autonomous mobile agents, object detection enables obstacle avoidance and tracking, path planning, and scene understanding, contributing to safe and efficient navigation. Similarly, in augmented and virtual reality devices, such as head-mounted displays, lightweight and energy-efficient object detection algorithms are crucial to enable real-time visual recognition without exceeding the limited power resources of the wearable device. A majority of recent research on object detection focuses on perfecting detection accuracy. State-of-the-art models have on the order of 10^8 or more parameters, requiring power-hungry hardware such as GPUs [9], [10] for inference. Therefore, state-of-the-art models like the ''You Only Look Once'' (YOLO) series of networks [11], [12], [13], [14], [15], [16], [17], [18] cannot be directly ported to low power edge processors due to the memory and compute constraints of MCUs. Consequently, there is a growing demand to enable semantic image understanding on the edge using ultra-low power and constrained hardware. This shift has led to a surge of interest in various research areas, including architecture search, quantization techniques, and advanced inference engines tailored for resource-constrained devices [19], [20], [21], [22]. MCUs are now being equipped with novel open-source energy-efficient cores, such as RISC-V cores, parallel processing engines, dedicated hardware accelerators, and specialized co-processors aimed at enabling efficient execution of complex machine learning tasks [23], [24].

By combining these advancements in parallel processing, hardware accelerators, and quantization techniques, MCUs are now executing quite sophisticated machine learning (ML) models. However, deploying machine learning on MCUs is still far from trivial and mapping networks like YOLO for advanced visual tasks, beyond simple classification, is still at the boundary of feasibility.

This work presents an improved and cutting-edge TinyissimoYOLO network designed to push the boundaries and to achieve accurate and real-time object detection on MCUs at sub-millijoule inference energy consumption. By leveraging novel low-power processors, we move from TinyissimoYOLO [22]—initially capable of detecting 3 object classes within a very restricted dataset—and present an improved, highly flexible network architecture, flexible TinyissimoYOLO, shown in Fig. 1. Our changes to the TinyissimoYOLO network allow the use of higher-resolution images while further increasing the number of object classes to detect. Additionally, we extensively evaluate the new flexible TinyissimoYOLO on multiple resolutions and on different numbers of objects to detect. Furthermore, the network in our first study was designed specifically for the MAX78000 MCU. In this paper, we provide a more detailed evaluation of the proposed network for low-power microcontrollers and further assess its performance on a novel multi-core architecture. This difference can aid future research by providing insights into adapting the proposed network across various platforms. Additionally, the network capitalizes on the unique capabilities of the latest generation of MCU platforms, harnessing their computational capabilities to set a new state-of-the-art in energy-efficient object detection, with a memory footprint of below 1 MB. Through extensive evaluation and performance analysis, we demonstrate the potential of our network in enabling energy-efficient computer vision applications.

This research contributes to the advancement of energy-efficient processing on resource-constrained devices,

opening up new possibilities for a wide range of real-world applications. To support our contribution, we evaluate a complete pipeline for deploying accurate, lightweight quantized object detection on several novel MCUs, bridging the gap between state-of-the-art models and on-edge execution. We provide a detailed description for each part of our pipeline, relying as much as possible on open-source tools, with the hope of enabling others to successfully deploy object detection models on ultra-low-power edge systems.

To quantify the benefits of the proposed approach, we evaluate the performance and energy efficiency of our object detection pipeline on several MCUs, executing different TinyissimoYOLO networks on one or multiple general-purpose cores as well as specialized convolutional neural network (CNN) accelerators. We investigate and expand the TinyissimoYOLO network proposed by Moosmann et al. [22], a lightweight general multi-object detection network optimized for a single processor, by extending its flexibility. Depending on the target platform's capabilities, TinyissimoYOLO can be scaled to make the best use of the target hardware and offer optimal performance. We deploy TinyissimoYOLO on multiple microcontroller architectures: the GAP9 multi-core RISC-V MCU (GreenWaves Technologies), which features a hardware accelerator for CNNs, MAX78000 (Analog Devices), a multi-core platform with an energy efficient CNN accelerator, Apollo4b (Ambiq), the most power-efficient ARM Cortex-M4 core currently available, as well as the STM32L4R9 and STM32H7A3 from STMicroelectronics to establish a baseline with the most popular ARM Cortex-M4F and Cortex-M7 cores.

Furthermore, for GAP9, we map a Pareto front of operating points (core voltage and operating frequency) to evaluate the trade-off between latency and inference energy. In addition, we evaluate the per-layer inference power consumption and quantify the efficiency enhancements enabled by the integrated NE16 CNN accelerator.

Additionally, we explore several variations of TinyissimoYOLO, spanning the trade-off between prediction accuracy and resource consumption. We evaluate the detection performance for a range of image resolutions and different kernel sizes applied in the first layer. We report the size of the network for each modification and the detection accuracy. Our investigation demonstrates that, as expected, increased input resolution contributes to higher detection accuracy, but also emphasizes the increasing memory consumption.

The rest of this paper is organized as follows: Section II provides an overview of the works focusing on CNN optimization and deployment on microcontrollers. Section III details our deployment pipeline, from choosing the architecture and training, to quantization and porting to target hardware. Section IV specifies the experimental setup and reports the results, presenting an in-depth analysis of the trade-off between performance and energy efficiency. Furthermore, we compare our GAP9-deployed model against state-of-the-art deployments on different MCUs, focusing on latency, energy efficiency and inference efficiency (MACs/cycle). Lastly, Section V concludes our work.

II. RELATED WORK

In the past decade, deep learning approaches have revolutionized the field of image-based scene understanding, through object detection [6], [7] and semantic segmentation [25], [26]. Semantic understanding of objects in the environment is an essential capability for autonomous agents, for tasks such as localization [27], mapping [28], [29], [30], and navigation [31], [32]. You Only Look Once or commonly abbreviated as YOLO is one of the most popular and optimized deep learning algorithms used to perform real-time detection [33]. To effectively detect and track objects, YOLO uses a repurposed classifier or localization which is a model applied to an image at several locations and scales [33]. However, the majority of works require power-hungry hardware such as GPUs and are not suitable for deployment on low-power edge devices. Even some of the more resource-conscious approaches ([33], [34]) still require powerful hardware consuming multiple watts and requires several tens of megabytes to run inference at sensor-rate. Based on the recent literature, we observed that existing YOLO approaches tend to have high memory requirements, limiting their applicability on resource-constrained devices. To address this limitation, in this paper, we evaluate and optimize a novel and flexible lightweight algorithm inspired by YOLOv1. The TinyissimoYOLO [22] algorithm has been specifically designed to achieve optimal performance in terms of accuracy while keeping the memory requirements below 500 kB. However, the proposed algorithm was only trained using a restricted dataset with a limited number of objects per image. In this paper, we trained and evaluated the algorithm with all the images of the PascalVOC dataset [35]—considering no object limitations—while investigating different input image resolutions to achieve higher detection accuracy, a small network layer adjustment for efficient feature extraction, and different number of object classes to detect in the dataset for testing the generalization capability of the network. This investigation leverages the full potential of the small-sized network's ability to generalize for multiple detection classes while further presenting benchmarking results in terms of memory consumption and compute requirement of the latest commercial microcontroller hardware.

The deployment of image classification and object detection models on MCUs has garnered significant attention in recent years. Canepa et al. [36] propose a method for detecting specific objects in surveillance video frames using deep neural networks on an STM32 MCUs. Although they achieve high prediction accuracy, their slow inference rate (0.03 Hz) limits its suitability for real-time applications, and their power consumption of approximately 400 mW is relatively high. To enable semantic understanding on edge devices and small autonomous agents, the models must be small and efficient enough to be executed on low-memory (<1 MB) and

ultra-low-power platforms (<100 mW) at significantly higher inference rates.

This task requires a specialized workflow [2], composed of three main strategies. Firstly, exploiting the benefits of resource-aware neural architectures and possibly automatically search for such neural architectures [37], [38], [39], [40]. Secondly, pruning and quantization strategies [41], [42], as offered by commonly used deep learning frameworks such as PyTorch. Notable quantization frameworks targeting ultra-low-power hardware include TensorFlow Lite [20] and Microsoft NNI [21], and other academic platforms. And lastly, deployment on lightweight hardware using inference engines, which aim to improve data locality, memory usage, and spatiotemporal execution.

TinyML software suites [19], including the open-sourced TensorFlow Lite Micro [43], EdgeML [44] and CMSIS-NN [45], allow for deploying neural networks on MCUs and are mainly designed for ARM Cortex-M and as such less attractive for RISC-V based processors. Similarly, Wulfert et al. [46] present an object detection method for resource-limited systems, performing camera-based human detection directly on a small ESP32 MCUs. While they achieve a high inference rate of 12 Hz, their approach is limited to detecting a single class. Likewise, Palossi et al. [23] demonstrate real-time human tracking on a nano drone, mounted with GAP8, a RISC-V parallel platform from GreenWaves Technologies, but are constrained to a single class. Lamberti et al. [24] propose a specialized low-power Automatic License Plate Recognition system executed on GAP8 at an approximate frequency of 1 Hz.

In contrast with these prior works, our paper makes several novel contributions targeting a variety of promising platforms. Firstly, our proposed flexible lightweight algorithm ensures competitive accuracy while keeping the memory requirements low enough for deployment on MCUs. Secondly, we target specifically different low-power MCUs, which serve as the processing units for a wide range of edge devices. This diverse platform evaluation further highlights the versatility and robustness of our approach. Overall, this work addresses the limitations of existing methods and introduces a memory-efficient algorithm suitable for resource-constrained devices, opening up new possibilities for efficient and accurate object detection in real-world applications. Additionally, the evaluation of our proposed algorithm on different platforms, including ARM, RISC-V cores and hardware accelerators, provides valuable insights into the benefits and trade-offs associated with each hardware architecture. This comparative analysis allows us to identify the strengths and weaknesses of each platform, enabling us to make informed decisions based on the specific requirements of the application at hand. This analysis provides a comprehensive view of the various approaches, highlighting the benefits of parallel processing in the multi-core RISC-V processor and the efficiency gains achieved through hardware accelerators. Furthermore, it offers a valuable perspective on power consumption, latency, and scalability, that influence the choice of hardware for object detection tasks on resource-constrained devices.

Giordano et al. [47] benchmark a single architecture for image classification on several different platforms. Moss et al. [48] evaluate different image classification architectures on a single platform, MAX78000. Unlike these works, we describe the full deployment pipeline in the context of object detection, from architecture exploration to quantization and hardware-optimized implementation.

In the realm of efficient neural architecture design for MCUs, MCUNet [49] presents a framework that combines the lightweight inference engine (TinyEngine) with the efficient neural architecture (TinyNAS), enabling ImageNet-scale inference on MCUs. Building upon this, MCUNetV2 [50] introduces memory-efficient patch-based inference, further enhancing memory performance for image classification and object detection. However, while these works report peak memory consumption and multiply-accumulate (MAC) operations, they lack comprehensive power consumption measures or run-time evaluations. By exploring various variations of the TinyissimoYOLO [22] network and optimizing it to be deployable in different platforms, we provide a detailed analysis of the trade-offs between prediction accuracy, run-time, and power consumption across multiple deployment platforms. Notably, we demonstrate the feasibility of reducing the MAC operations to approximately 3 million, representing a 10-50 times reduction compared to MCUNetV2. In contrast to the previously published TinyissimoYOLO paper, which evaluated the network's performance on a restricted subset of the PascalVOC [35] dataset and only on single-core processors, this work investigates the network's performance without any restrictions on the dataset and explore and benchmark the flexibility and the hardware overall energy efficiency. This allows us to evaluate the performance of our flexible TinyissimoYOLO version across the entire PascalVOC dataset, considering all classes and unrestricted object counts within each image. By combining advancements in architecture design, memory-efficient inference, quantization-aware training, and a comprehensive evaluation across various deployment platforms, our work makes significant contributions to the field. We showcase the remarkable reduction in MAC operations achieved by our proposed approach, surpassing the state-of-the-art MCUNetV2. Furthermore, our thorough evaluation on the complete PascalVOC dataset demonstrates the robustness and scalability of our flexible TinyissimoYOLO network. Overall, our work pushes the boundaries of energy-efficient object detection on MCUs, providing valuable insights and paving the way for further advancements in the field.

III. BACKGROUND AND IMPLEMENTATION

In this section, we describe *TinyissimoYOLO*, which we used as the basis for our explorations, training, and dataset, the

MCU platforms we compare to and deploy on, and the deployment tools.

A. TINYISSIMOYOLO

TinyissimoYOLO is a general multi-object detection network, designed to enable fast and accurate detection on microcontroller platforms. Its architecture is shown in Fig. 1. The original network, as proposed by Moosmann et al. [22], uses an input resolution of 88 × 88 pixels and produces an output vector of dimension (S × S (B ∗ 5 + C)), where S × S is the grid of prediction cells, B is the number of boxes predicted per cell and C is the number of classes. This work explores the flexibility of this network by: increasing the input resolution to trades improved detection performance for increased computational load and a proportional increase in the number of parameters of the last layer. The number of detected classes C can also be varied, affecting only the parameter count and computational volume in the last layer. The CNN backbone used for feature extraction is small in comparison to state-of-the-art object detection networks. The original YOLOv1 network has 20 GMAC and 45 M parameters, while the more recent YOLOv7 scales from 1.75 GMACs to 420 GMACs and 6.2 M to 151 M parameters. In comparison, the deployed TinyissimoYOLO can be scaled from 32 MMACs to 57 MMACs and from 441 K to 887 K parameters. Using 8-bit quantization further reduces the model size and memory footprint by a factor of 4 when compared to equivalent 32-bit floating-point models, making TinyissimoYOLO ideally suited for real-time inference on resource-constrained MCU platforms.

In this work, we investigate the influence of network parametrization on detection performance. In contrast to the initially proposed TinyissimoYOLO [22], we train TinyissimoYOLO with different input resolutions (88 × 88, 112 × 112 and 224 × 224 pixels), different numbers of output classes C (3, 10 and 20) and different kernel sizes in the first layer (3 × 3 and 7 × 7). Further, no dataset limitations are posed, and no limitations to the number of objects per image are set, which is in contrast to the initially proposed TinyissimoYOLO network.

B. DATASET

Since the number of network parameters in the output layer increases linearly with the number of object classes in the dataset—Fig. 1 shows the output layer is defined by S × S (B ∗ 5 + C)—we chose the PascalVOC [35] dataset for investigating the networks' detection accuracy. Choosing another dataset with more diverse classes would lead to even bigger TinyissimoYOLO networks as can be seen in Table 2 when comparing the networks' number of parameters with the number of classes. 90 % of the PascalVOC training dataset was used to train the network, with the remaining 10 % serving as the validation set. The training data was augmented with geometric operations such as cropping, scaling, and shifting, as well as photometric operations including blurring, and modifying the brightness, contrast, saturation, and hue.

C. NETWORK TRAINING AND QUANTIZATION

The hardware platforms we target offer the best performance and full toolchain support for networks quantized to 8-bit weight and activation precision. While smaller bitwidths can offer further memory and storage savings [52], the lack of native hardware support for sub-byte arithmetic would lead to significant implementation and runtime overheads with such aggressively quantized networks [53]. Furthermore, quantization to sub-byte precision often incurs substantial accuracy degradation. For these reasons, we evaluate only 8-bit quantized networks [54]. We used *QuantLab*[1] [55] to train and quantize the evaluated networks. QuantLab is a modular, open-source framework for quantization-aware training (QAT) based on PyTorch, offering experiment management facilities, support for various quantization algorithms, and automated model conversion functionality from full-precision to trainable (*fake-quantized*) and deployable integer-only quantized models. We train the 8-bit TinyissimoYOLO networks in two phases, which process is performed using Nvidia RTX4090 GPUs: First, a full-precision network is trained to convergence. In the second step, we perform QAT using the TQT [56] algorithm. QuantLab converts the original architecture to a fake-quantized version,[2] where each convolution, fully connected and activation layer is replaced with its quantized equivalent. We initialize the fake-quantized networks from the corresponding full-precision checkpoint, training with weight-only quantization for some epochs before enabling full-model quantization of weights and activations. The full hyperparameters for both training phases are listed in Table 3. After QAT has converged, the trained fake-quantized model can be automatically converted to an integer-only model. In the integerized model, normalization, rescaling and activation layers are merged into requantization layers. A requantization layer consists of channel-wise integer multiplication, channel-wise addition, logical right shift and clipping, effectively executing an affine transformation and clipping in fixed-point arithmetic. This approach has been described multiple times in literature and has been variously termed ''integer channel normalization'' [52] or ''dyadic quantization'' [58]. Finally, the integerized model is exported as a backend- and hardware-agnostic ONNX model, where the exported ONNX operators are annotated with precision information, allowing the deployment backend to select the correct kernels. Both the fake-quantized TinyissimoYOLO models trained in QuantLab and the deployable integer-only models generated from them exhibited no accuracy drop compared to their full-precision counterparts. The integer models generated this way are used for our accuracy evaluations in Table 2. Only the GAP9 cluster deployment pipeline supports processing the produced ONNX files.

[1] https://github.com/pulp-platform/quantlab

[2] A fake-quantized network takes values in a discrete subset of $\mathbb{R}$. After converting it to its fully integerized (or *true-quantized*) counterpart, it can be executed using integer arithmetic [57].

TABLE 1. TinyissimoYOLO network trained and evaluated on PascalVOC with different network configurations. This table shows the network performance for the different network configurations. The naming convention of the different network configurations is: TY(TinyissimoYOLO):classes-1st layer's kernel-input resolution, for more details, see: Table 2.

Network	mAP																				
Arch.	*mAP*	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
TY:3-3-88	62%							70%		45%						70%					
TY:3-7-88	62%							71%		45%						69%					
TY:3-3-112	63%							73%		46%						70%					
TY:3-7-112	62%							72%		46%						68%					
TY:3-3-224	68%							74%		53%						77%					
TY:3-7-224	68%							73%		54%						76%					
TY:10-3-88	58%	70%	66%	51%	52%	28%	72%	69%	69%	44%	62%										
TY:10-7-88	58%	68%	67%	54%	54%	27%	70%	69%	69%	46%	60%										
TY:10-3-112	60%	71%	71%	54%	57%	30%	73%	73%	69%	47%	59%										
TY:10-7-112	57%	68%	67%	50%	50%	22%	71%	69%	68%	45%	60%										
TY:10-3-224	65%	77%	74%	57%	63%	39%	74%	76%	72%	49%	66%										
TY:10-7-224	54%	65%	63%	47%	50%	26%	64%	67%	65%	41%	49%										
TY:20-3-88	53%	72%	68%	53%	52%	0%	70%	71%	70%	38%	44%	65%	66%	73%	67%	65%	5%	55%	9%	69%	48%
TY:20-7-88	47%	67%	60%	42%	46%	0%	70%	68%	66%	36%	12%	63%	59%	66%	66%	64%	0%	55%	24%	65%	49%
TY:20-3-112	56%	71%	70%	54%	54%	4%	71%	73%	69%	39%	38%	64%	66%	74%	68%	66%	23%	62%	36%	71%	56%
TY:20-7-112	53%	72%	64%	53%	55%	0%	71%	71%	67%	37%	34%	63%	62%	74%	67%	65%	5%	53%	35%	68%	53%
TY:20-3-224	60%	71%	67%	55%	56%	14%	71%	72%	70%	44%	51%	60%	67%	72%	67%	63%	39%	62%	60%	70%	61%
TY:20-7-224	67%	75%	72%	61%	64%	35%	75%	77%	76%	48%	66%	68%	72%	76%	72%	71%	44%	64%	73%	76%	69%
YOLOv1 20-7-448	73%	82%	76%	71%	67%	45%	82%	82%	82%	54%	72%	72%	79%	81%	76%	76%	51%	71%	76%	81%	77%

TABLE 2. TinyissimoYOLO network trained and evaluated on PascalVOC. This table shows the different network configurations which are evaluated. Additionally, the number of network parameters and the corresponding network model sizes are shown.

Network	# classes	1st layer's kernel	input res.	net. param.	model Size	MMACs	mAP
TY:3-3-88	3	3	88 × 88	440,592	441 KiB	**32**	61.8%
TY:3-7-88	3	7	88 × 88	442,512	443 KiB	47	61.5%
TY:3-3-112	3	3	112 × 112	573,712	574 KiB	55	63.1%
TY:3-7-112	3	7	112 × 112	575,632	576 KiB	79	61.9%
TY:3-3-224	3	3	224 × 224	1,638,672	1.64 MiB	220	68.0%
TY:3-7-224	3	7	224 × 224	1,657,872	1.66 MiB	316	67.8%
TY:10-3-88	10	3	88 × 88	498,048	498 KiB	33	58.3%
TY:10-7-88	10	7	88 × 88	499,968	500 KiB	47	58.4%
TY:10-3-112	10	3	112 × 112	702,848	703 KiB	55	60.4%
TY:10-7-112	10	7	112 × 112	722,048	722 KiB	79	56.9%
TY:10-3-224	10	3	224 × 224	2,341,248	2.34 MiB	220	64.8%
TY:10-7-224	10	7	224 × 224	2,360,448	2.36 MiB	317	53.8%
TY:20-3-88	20	3	88 × 88	580,128	580 KiB	33	53.1%
TY:20-7-88	20	7	88 × 88	582,048	582 KiB	47	47.0%
TY:20-3-112	20	3	112 × 112	887,328	887 KiB	55	56.4%
TY:20-7-112	20	7	112 × 112	906,528	907 KiB	79	53.5%
TY:20-3-224	20	3	224 × 224	3,344,928	3.34 MiB	221	59.5%
TY:20-7-224	20	7	224 × 224	3,346,848	3.35 MiB	318	66.6%
MbV2+CMSIS [49]	20		128 × 128	0.87M	0.87 MiB	34	31.6%
MCUNet [49]	20		224 × 224	1.2 M	1.2 MiB	168	51.4%
MCUNetV2-M4 [51]	20		224 × 224	1.01 M	1.01 MiB	172	64.6%
MCUNetV2-H7 [51]	20		224 × 224	2.03 M	2.03 MiB	343	**68.3%**

As the aim of this paper is a separate evaluation of platform efficiency and model performance, we only deploy the fully trained and quantized models on GAP9's cluster. For all other targets, we use the platform-specific quantization and deployment flows on untrained networks and without evaluating the resulting models' accuracy. However, it is important to note that all of the evaluated platforms support the execution of this or an equivalent class of integerized models, meaning that our results are representative for all platforms.

TABLE 3. Training hyperparameters for full-precision training and quantization-aware training (QAT) of our networks. We performed QAT using the trained quantization thresholds (TQT) algorithm.

Phase	Full-Precision	QAT
Batch Size	64	64
# Epochs [a]	1000/7000	500/1000
Optimizer	SGD	SGD
LR Sched.	Const. Sched.	Const. Sched.
LR_0	$5 * 10^{-4}$	$5 * 10^{-4}$
Wt. Quant.[b]	N/A	+5
Act. Quant.[c]	N/A	+50

[a] Networks with 88×88 and 112×112 input image resolution are trained 1000 epochs on full precision and 500 epochs QAT, while networks with 224×224 input image resolution are trained with 7000 epochs and 1000 epochs QAT.
[b] Training epoch in which weight quantization is started.
[c] Training epoch in which activation quantization is started.

D. MCU PLATFORMS

We compare deployments on different MCU platforms, which we introduce here.

1) ARM CORTEX-M4 AND CORTEX-M7

The used MCUs from STMicroelectronics (STM32H7A3 and STM32L4R9) each feature an ARM Cortex-M single-core processor. The H7A3 and L4R9 MCUs use a Cortex-M7 and a Cortex-M4 operating at up to 280MHz and 120MHz respectively, with core voltages of up to 1.3V.

Apollo4b also uses an ARM Cortex-M4 processor running up to 192MHz with a nominal core voltage of 0.65V. Apollo4b's main distinguishing characteristic

is its utilization of Ambiq's proprietary subthreshold power-optimized technology platform, designed to offer maximum power efficiency for edge applications.

MAX78000 features an ARM Cortex-M4, a built-in CNN accelerator which has 64 specialized processors with built-in convolutional engine, pooling unit and dedicated 442 kB weight memory. In addition, a 32-bit RISC-V coprocessor supports ultra-low-power signal processing.

2) GAP9

GAP9 features 10 RISC-V cores. One core acts as a *fabric controller*, orchestrating system operation, while a parallel ultra-low-power (PULP) cluster of 9 cores implementing custom instruction set extensions is available for efficient, high-performance execution of compute-intensive tasks. Additionally, it includes NE16, a dedicated on-chip hardware accelerator for CNN inference. The GAP9 architecture is based on the open-source system on chip (SoC) Vega [59]. The cores' maximum operating frequency is 370 MHz for both the 9-core cluster and the fabric controller. For additional flexibility, GAP9 is provisioned for dynamic voltage and frequency scaling, allowing users to trade-off between latency and energy efficiency. The compute cluster, consisting of 9 cores, one for orchestration and 8 workers, offers general-purpose compute power at extreme energy efficiency while the CNN hardware accelerator NE16 (based on RBE [60]) is specialized for highly efficient MAC operations. NE16 features 9×9x16 8×1bit MAC units, which are optimally used in 3×3 convolutions, but it also offers support for 1×1 and 3×3 depth-wise convolutions and fully connected layers. GAP9 has a hierarchical memory layout, with 128 KiB of high-bandwidth, single-cycle-accessible L1 scratchpad memory in the cluster, 1.5 MiB of interleaved L2 memory for data and code as well as 2 MiB of on-chip flash memory. GAP9 also offers a rich set of peripherals for connecting to external memory, sensors, and standard interfaces such as UART, I^2C, CSI, and others.

E. MCU DEPLOYMENT

Different MCU platforms require different deployment tools, which we introduce in this section. Note that we deploy networks to two compute domains on GAP9 (the RISC-V cluster and the NE16 CNN accelerator) using two different flows as described below and depicted in Fig. 2.

1) GENERAL-PURPOSE PROCESSOR DEPLOYMENT

For deployment to GAP9's cluster, we use the DORY framework [61] to map the precision-annotated ONNX files generated by QuantLab. DORY is an automated deployment utility for ultra-low-power edge platforms with hierarchical memory layouts. It takes a precision-annotated ONNX file as the input and generates ANSI C code which implements the specified network on the target platform. Tiling between up to three hierarchical memory levels (L1 scratchpad memory, L2 main on-chip memory and L3 off-chip memory) is automatically performed with an integer linear programming (ILP)-based tiling algorithm which takes into account the hardware-specific constraints (i.e., the memory size of each hierarchical level) and various heuristics.

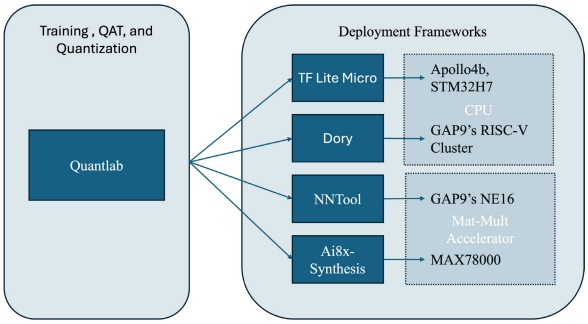

FIGURE 2. Visualizes the training and the deployment workflow with the corresponding tools used for each device.

For deployment on the ARM Cortex-M4/M7 platforms, we use TensorFlow-Lite Micro [43]. As such, the deployment on the single-core ARM Cortex-M4 and M7 is performed by quantizing the network weights to 8-bits, generating the C++ code using TensorFlow-Lite Micro, and compiling the code for the corresponding microcontroller.

2) CNN ACCELERATOR DEPLOYMENTS

The deployment on MAX78000's CNN accelerator was performed by using Analog Devices' ai8x-synthesis framework to quantize the network weights, activations, and input using a forked version of the Neural Network Distiller by Intel AI Lab.[3] Finally, the C code used to deploy the network on the MAX78000's CNN accelerator is generated by the ''izer'' tool, which converts the quantized trained model into C code. The complete network fits the accelerator and all the weights can be stored inside the weight memory of the accelerator.

Deployment of the network on GAP9's NE16 neural engine was done in collaboration with GreenWaves Technologies. To deploy networks on the GAP9 microcontroller, GreenWaves Technologies distributes a deployment framework called NNTool as part of the GAP SDK.[4] NNTool is used for 1) post-training network quantization 2) network evaluation for activation and parameter sizing and 3) code generation for deployment. Analogous to DORY, it calculates a tiling of the model's individual layers such that the data for each tile fits into the L1 scratchpad.

3) END-TO-END EVALUATION

To perform an accurate measurement including the image acquisition, we implemented the complete sensing pipeline consisting of the microcontroller platforms running TinyissimoYOLO and attached RGB cameras. In this way, the edge inference can be performed with real-world data, without relying on synthetic data from the dataset. We attached an OV5647 RGB CMOS camera from Omnivision to the GAP9 and attached the OVM7692 CameraCubeChip to

[3]https://nervanasystems.github.io/distiller
[4]https://github.com/GreenWaves-Technologies/gap_sdk

the MAX78000. As such we are able to demonstrate the functionality of the trained networks on two different devices. We performed our measurements on the GAP9 evaluation Kit and on a custom-designed PCB for the MAX78000, shown in Fig. 3.

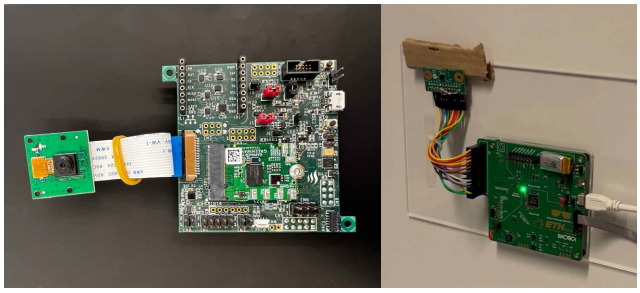

FIGURE 3. Left: GAP9 evaluation board with the OV5647 camera attached. Right: custom-developed PCB featuring the MAX78000 with the OVM7692 camera attached.

Further, Fig. 4 shows some examples of images captured on GAP9 after processing with the TinyissimoYOLO network, while Fig. 5 shows some other example images of a demo running image capturing and inference on the MAX78000, while streaming the image (after adding the detected boxes on the MAX78000 itself) live, via UART, to an attached PC.

FIGURE 4. Example recordings of images with GAP9 and running an inference of TinyissimoYOLO on the recorded images.

IV. EXPERIMENTAL RESULTS

To evaluate the trade-off between detection performance and energy efficiency, we trained eighteen different network variants on the PascalVOC dataset and report the mean average precision (mAP) for each class in Table 1. All network versions shown—with up to 20 detection classes and input image resolutions of 112 × 112—can be deployed on general-purpose MCUs such as GAP9. To compare the performance of GAP9 with the reported performance on the paper [22], we deployed TinyissimoYOLO (TY:3-3-88) on GAP9. First, we deployed it on single-core MCUs only[5] to fairly compare to other single-core implementations such as STM32H7A3 featuring an ARM Cortex-M7,

[5]Despite GAP9 being multi-core, the network deployment was done such that the inference runs only on one of the nine cluster cores.

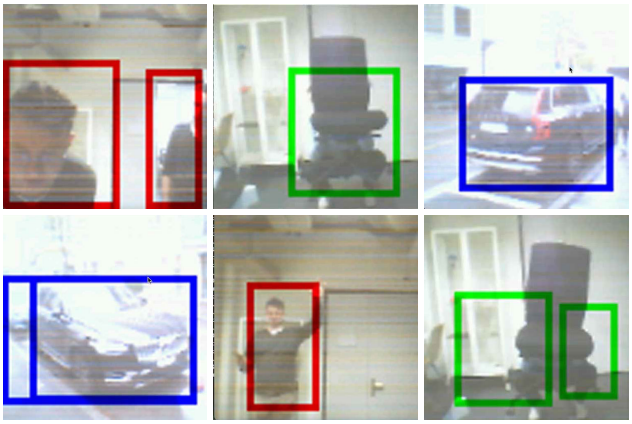

FIGURE 5. Example recordings of images with MAX78000 running an inference of TinyissimoYOLO on the recorded images.

STM32L4R9 with an ARM Cortex-M4 and the sub-threshold computing microcontroller Apollo4b from Ambiq, also with a single-core ARM Cortex-M4. Then we also deployed a parallelized implementation, for the best possible performance on the eight-core RISC-V cluster of GAP9 without making use of the built-in neural engine hardware accelerator, showing the advantages of a parallel platform. Using GAP9's capability to set the core voltage and frequency, we provide an analysis of different operating points, mapping a Pareto front of the latency-efficiency trade-off. As a last step, the network is deployed on the NE16 neural engine of GAP9.

TinyissimoYOLO was limited by the 442 kB in accelerator weight-memory of the MAX78000, which restricted it to a network with only 3 out of 20 available detection classes being trained, with an input resolution of 88 × 88 pixels to avoid relying on a specialized input streaming mode. As GAP9 features a bigger (1.5 MB) on-chip memory and can even use external memory, we also chose a second variant of the network, based on the results gathered. The second version features an input resolution of 112 × 112 pixels and is trained to detect 10 detection classes and has been deployed on GAP9 (TY:10-3-112) in a single-core, multi-core, and neural engine accelerated version. We compare our implementations on GAP9 with the deployment on the MAX78000 MCU. The single-core, multi-core, and neural engine deployment performances are provided. We measured the power consumption of the whole SoC which we supplied with 1.8 V and toggled a GPIO to detect the start and end of the CNN execution.

A. NETWORK ARCHITECTURE VARIATIONS

The 18 trained networks vary in terms of network image input resolution, the first layer's kernel size, and the number of classes the network is trained for object detection. Notably, increasing the input resolution yields an increase in the input to the last fully connected layer, while increasing the number of classes yields an increase in the output of the last fully connected layer. In particular, by changing the input or the number of classes to detect, the number of network weight parameters will change accordingly. Therefore, Table 2 lists

all the trained variants of TinyissimoYOLO and reports the number of parameters and the memory required to store all the network weights in quantized 8-bits, and the mAP. Table 2 further compares the result to the networks reported in Lin et al. [49] (MBv2+CMSIS and MCUNet) and Lin et al. [51] (MCUNetV2-M4 and MCUNetV2-H7). While the TiniyssimoYOLO networks which are also trained on images with 224×224 resolution have much more parameters compared to the MCUNets version one and two, they achieve a lower mAP. However, TinyissimoYOLO with an input resolution of 112 × 112 has fewer parameters (about the same as MBv2+CMSIS) but achieves higher mAP than MCUNetv1. Additionally, MCUNet has 168 MMACs with a detection accuracy of 51.4% (MCUNetV2-M4 has 172 MMACs and MCUNetV2-H7 has 343 MMACs), while the largest deployable TinyissimoYOLO (TY:20-7-112) only has 57 MMACs achieving higher detection accuracy of 56.4% mAP. Therefore, the small resolution TinyissimoYOLO version has a great trade-off between detection accuracy, network simplicity, and deployability onto accelerated MCU hardware. Table 1 reports the mAP for each detection class as well as the network's overall mAP achieved by the quantized to 8-bit networks. We varied the network input resolution between 88 × 88 pixels, 112 × 112 pixels, and 224 × 224 pixels. Even though we report training the network with an input resolution of 224 × 224 pixels, the network learning rate starting with 0.001 is unstable at the beginning leading the network to not get trained properly. However, by setting the initial learning rate to 0.0001, the network training is stable again, even though it takes more epochs to be trained.

When varying the number of object classes to predict, from 3 classes to 20 classes, for 88 × 88 pixel resolution the number of parameters scales by a factor of 1.3×, while for 224 × 224 pixel resolution, the scaling factor is 2×. Notably, changing the network's first kernel to a kernel size of 3 × 3 to 7 × 7, we unsurprisingly note a constant increase of 1920 parameters.

Comparing the performance of the various TinyissimoYOLO networks, we first report the change of the first layer's kernel size from 3 × 3 to 7 × 7, which increases the detection accuracy to 67% mAP for the network TY:20-7-224. Compared to the network TY:20-3-224, this is an increase of +7%. However, for networks of less than three million network parameters, the kernel change leads to no improvements for 3 detection classes, while it decreases the detection accuracy for all remaining networks. Increasing the input resolution constantly increases the mAP performance in overall network detection accuracy. Notably, decreasing the number of detection classes increases the overall mAP performance of the network and of each detection class itself.

B. GAP9 - RISC-V MCU PERFORMANCE

We deployed two different CNNs, the original TinyissimoYOLO (TY:3-3-88) for a fair comparison and an adapted network (TY:10-3-112) for more classes and higher accuracy.

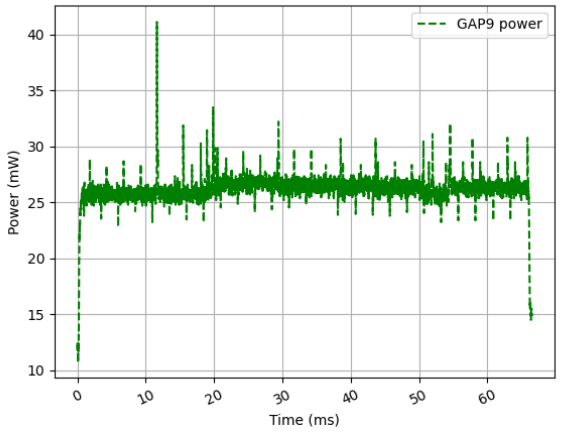

FIGURE 6. TY:3-3-88 single-core execution consumes 26.14 mW on average over 69.77 ms, resulting in an energy consumption of 1738 µJ.

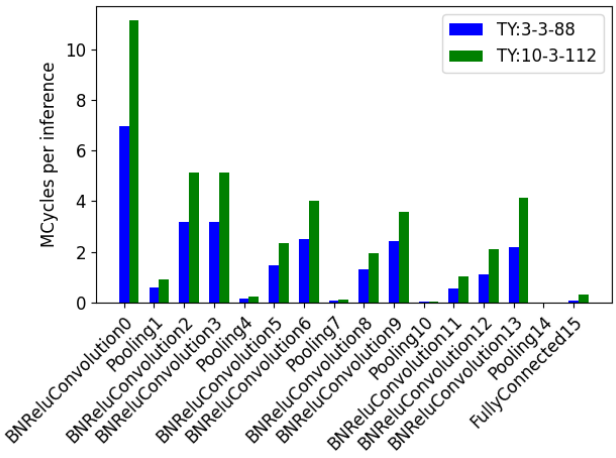

FIGURE 7. Single-core execution of TY:3-3-88 (in blue) achieves 1.25 MAC/cycle and accumulates to a total of 26 Mcycles, distributed to the different layers as shown here. Single-core execution of TY:10-3-112 (in green) achieves 1.29 MAC/cycle and accumulates to a total of 42 Mcycles, distributed to the different layers as shown here.

1) SINGLE-CORE PERFORMANCE

Here we report the single-core performance of both networks deployed on GAP9.

a: TY:3-3-88

Single-core execution on GAP9 results in 26 Mcycles, so an equivalent of 69.77 ms at the maximum frequency of 370 MHz while we reach 1.25 MAC/cycles. The average power consumption is 26.14 mW, which gives us an energy consumption of 1738 µJ. In Fig. 6 we show the power consumption of the single-core implementation, showing a stable power consumption with a ripple, possibly from the internal voltage regulator as we measure the whole SoC power. In Fig. 7 we show the number of cycles by layer in blue - we see that the convolutional layers are the most computationally expensive, especially the first one.

b: 112×112 INPUT 10 CLASSES

The single-core execution time of this network is 114.15 ms while consuming 2990 µJ per inference. We show the cycles per layer in Fig. 7 in green, seeing again that the first layer consumes most cycles and max-pooling is insignificant with only 3% of the overall number of cycles.

2) GAP9 MULTI-CORE MCU PERFORMANCE

We deployed two different CNNs, first the original TinyissimoYOLO for a fair comparison and then an adapted network for more classes and higher accuracy. Here we report the multi-core performance of both networks.

a: TY:3-3-88

In Fig. 8 we show the speedup for execution on 8 versus 1 core, for parallelizing by columns and, for the convolutional layers, by output channels. We first parallelized by columns, which gives us good results for the first layers, however, leads to low speedup on small spatial dimensions of feature maps. Therefore we switched to parallelizing by output channels for layers 11 and 12, gaining 245k cycles. Layer 13 can not be parallelized by output channels, as it requires a higher stack size than what we can allocate.

Overall we achieve a speedup of 6.14x running on 8 versus 1 core, only needing 4.4 MCycles.

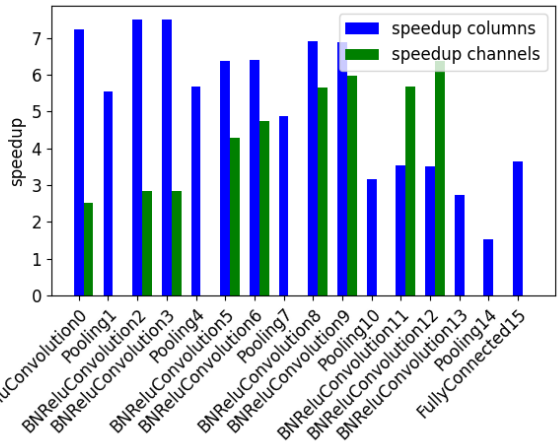

FIGURE 8. TY:3-3-88 speedup from 1 to 9 cores per layer for the two different parallelization schemes for convolutional layers.

GAP9 can run at frequencies up to 370 MHz on both the fabric controller and the cluster while maintaining an inference efficiency of 7.73 MAC/cycle on the multi-core cluster. We measured energy consumption and latency for different operating points between 50 MHz and 370 MHz, always choosing the minimum core voltage at which the system is still able to operate (in 50 mV steps). The MAC/cycle is not dependent on the frequency, meaning the latency scales linearly with the frequency.

In Fig. 9 we show our results, marking Pareto front points in green. We reach the most energy-efficient point at 150 MHz, as this allows operation at the minimum voltage of 0.65 V. At this operating point, GAP9 only consumes

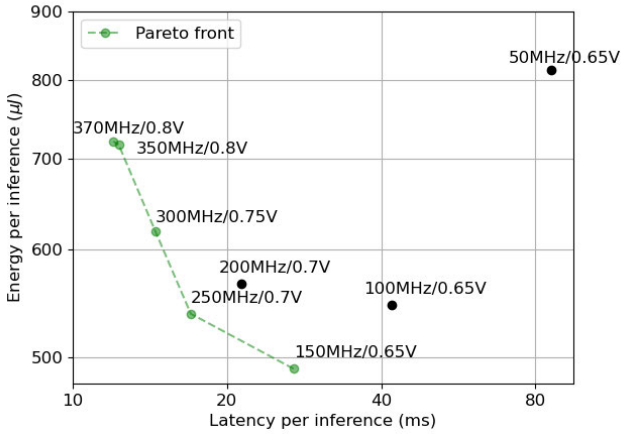

FIGURE 9. Latency versus energy efficiency at different operating points, showing the Pareto optimal set in green.

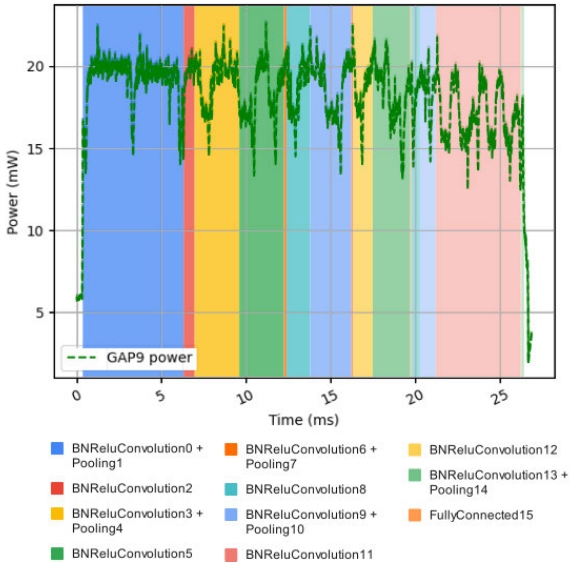

FIGURE 10. Energy-efficient TinyissimoYOLO on GAP9 has a latency of 27.87 ms and an average power consumption of 18.16 mW.

490.21 µJ per inference and exhibits a latency of 27.9 ms. Peak performance is reached at the maximum operating frequency of 370 MHz, with only 11.3 ms latency and an inference energy of 721 µJ.

In Fig. 10, we show the power consumption per layer for the most energy-efficient operating point (150 MHz at 0.65 V). We see the resulting latency, 27.87 ms, which corresponds to an energy per inference of 490 µJ. In Fig. 11 we show the power consumption per layer for the least latency operating point, the aforementioned 370 MHz at 0.8 V. We see the resulting latency, 11.3 ms, which with the measured energy efficiency of 162 µW/MHz corresponds to an energy per inference of 721 µJ.

In both Fig. 10 and Fig. 11 we visualize the execution times of the different layers by different colors. We can note a tendency that layers that parallelize better consume more power - which is expected, as the cluster then is fully used.

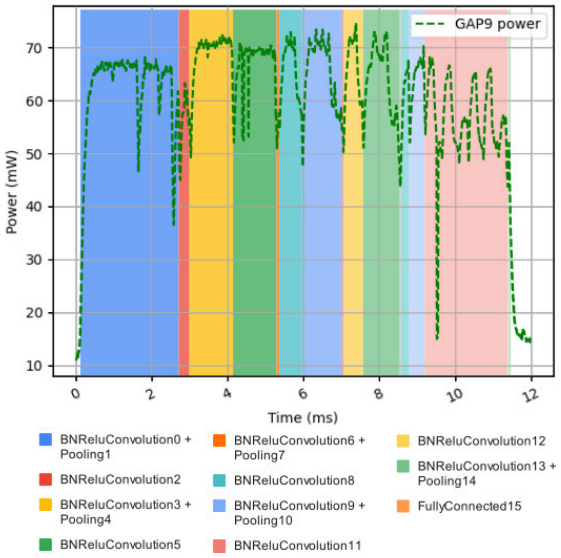

FIGURE 11. Peak performance TinyissimoYOLO on GAP9 has a latency of 11.3 ms and an average power consumption of 55.76 mW.

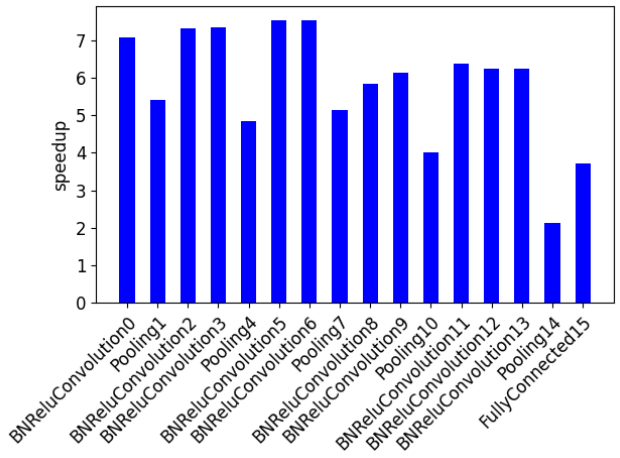

FIGURE 12. TY:10-3-112 speedup from 1 to 9 cores per layer, resulting in an average speedup of 6.77 and 8.74 MAC/cycle.

b: TY:10-3-112

We achieve an overall speedup of 6.77 when parallelizing on 9 cores. In Fig. 12 we show the speedup per layer, as before earlier convolutional layers parallelize better due to the higher number of columns - however, in this network, there is no need to parallelize by output channels as the speedup of the last convolutional layers is still above 6. At peak performance (370 MHz) we achieve a latency of 16.87 ms while consuming 1057 µJ per inference. Optimizing for energy-efficiency and running at 150 MHz, latency increases to 41.62 ms while consuming 765 µJ.

c: COMPARISON TO ARM

In Fig. 13 we compare our deployment on GAP9 RISC-V cores (single-core as well as multi-core) against deployments on the H7A3, L4R9, and Apollo. Comparing the single-core versions, GAP9 clearly outperforms the other architectures in terms of latency (by a factor of more than 10 to the next best

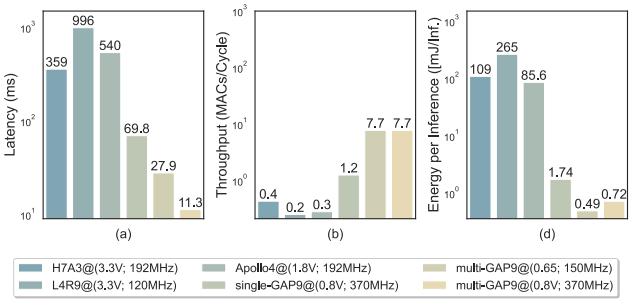

FIGURE 13. TY:3-3-88 performance comparison when deployed quantized to 8-bit on different MCU architectures. The GAP9 single- and multi-core implementation outperforms the other architectures in terms of latency, inference efficiency, and energy per inference.

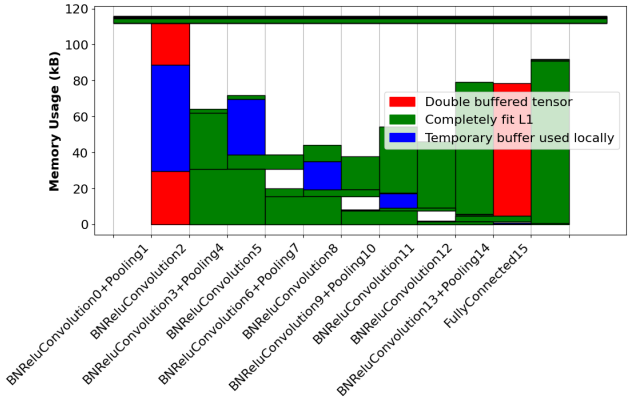

FIGURE 14. L1 memory allocation TY:3-3-88: The autotiler tiles weights, in and output layers for copying them into L1 where necessary. However, most layers are small enough to not require double buffering.

architecture) and energy per inference (by a factor of almost 50). In inference efficiency, we can also outperform the other architectures, as we can execute vectorized 8-bit operations on GAP9 contrary to the other architectures. For our multi-core implementation, we show the most energy efficient (150 MHz) and least latency (370 MHz) operating points of GAP9. Compared to the peak performance single-core implementation we can either achieve similar latency and around 3x reduced energy per inference or reduce latency by a factor of around 2.5 but only be around 2x more energy-efficient.

C. GAP9 HARDWARE ACCELERATOR PERFORMANCE

We also deployed the proposed two networks on the Hardware accelerator on GAP9.

1) TY:3-3-88

In Fig. 15 we show the cycles and MAC/cycle per layer. We have an array of 9 × 9 NE16 engines that can handle 16 multiplications at a time, so are ideal for a multiple of 16 input channels. Note that max-pool layers can not be executed on the accelerator, but have to be computed on the cluster. Those two factors compromise the MAC/cycle number for the first layer. On average we achieved 41.22 MAC/cycle. As the weights are constantly kept in L2, the L2 requirements are almost constant, only

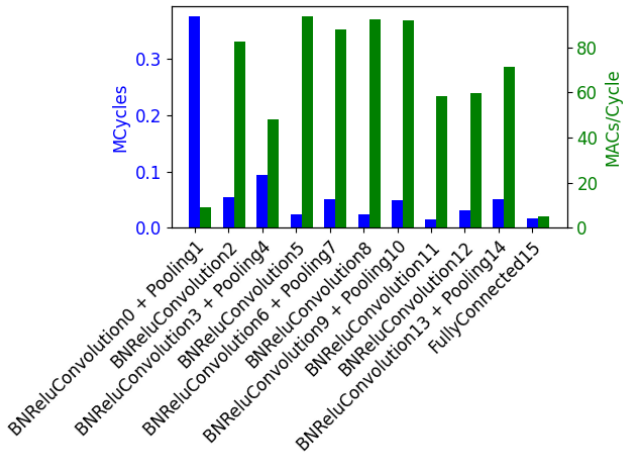

FIGURE 15. Using the HW accelerator on GAP9 we can reach an average of 41.22 MAC/cycle for TY:3-3-88, which leads to a total of 785 kcycles.

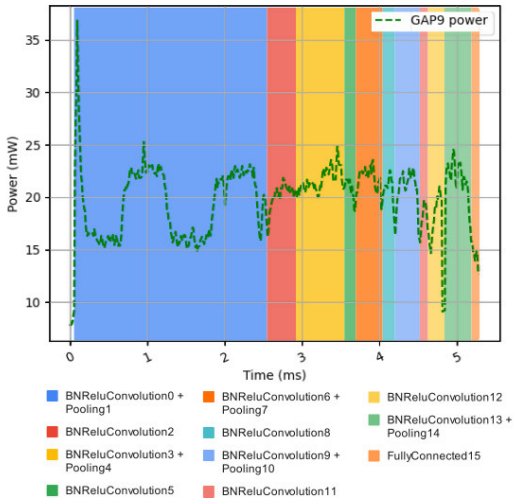

FIGURE 16. Energy-efficient TY:3-3-88 on the GAP9 hardware accelerator has a latency of 5.24 ms and an average power consumption of 20.04 mW.

the first convolution needs 31kB of additional dynamic L2 memory for double buffering. The L1 requirements per layer are displayed in Fig. 14. In Fig. 16 (150 MHz, the most energy-efficient operating point) and Fig. 17 (370 MHz, the least latency operating point) we show the power consumption for the network running on the GAP9 hardware accelerator. We observe that while the hardware accelerator is active the power consumption is higher than when only the cluster is active (on max-pooling operations, which is executed tiled in two parts on the first layer). At (370 MHz we achieved a latency of 2.12 ms and an energy per inference of 149 µJ, which is a 5.3x speedup compared to only using the general-purpose cores. At 150 MHz the latency is 5.24 ms while the energy per inference is 105 µJ, reducing the energy by 79% compared to the multi-core implementation.

2) TY:10-3-112

Deploying the 112×112 input network on the HW accelerator at 370 MHz we reached 42.84 MAC/cycle and a latency of 3.46 ms, while consuming only 245 µJ. Compared to the

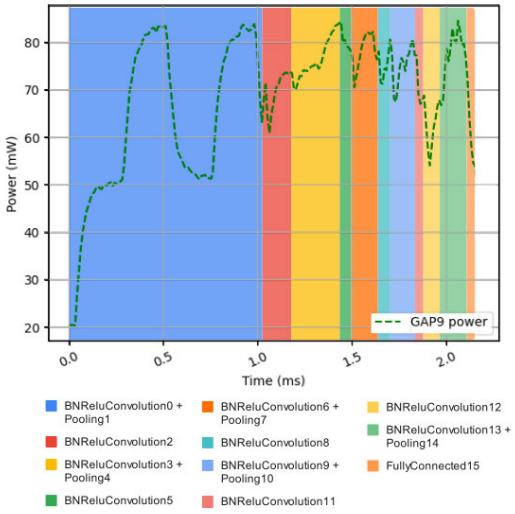

FIGURE 17. Peak performance TY:3-3-88 on the GAP9 hardware accelerator has a latency of 2.12 ms and an average power consumption of 70.30 mW.

multi-core implementation this is a 4.9x speedup. Running at 150 MHz we can improve the energy efficiency to 177 µJ per inference while increasing the latency to 8.54 ms, reducing the energy per inference by 77% compared to the multi-core implementation.

a: COMPARISON TO MAX78000

In Fig. 18 we compare the most energy efficient (150 MHz) and least latency (370 MHz) operating points of GAP9 to the implementation on the CNN accelerated MAX78000 MCU. Experimental results show the MAX78000 outperforms the single-core and multi-core implementations on GAP9 in terms of latency, inference efficiency, and energy per inference. However, the network implementation on the neural engine of GAP9 outperforms the inference latency and energy per inference of the MAX78000 because of the high clock frequency available. Even though the inference efficiency is 2.47x times less with 43.2 MAC/cycle, GAP9 with the NE16 reaches a latency of only 2.12 ms and energy consumption of 149 µJ at peak performance, being 2.6x faster and 1.3x more energy efficient than the MAX78000. At the most energy efficient frequency GAP9 still reaches a slightly lower latency than the MAX78000, but can even reduce the energy consumption by a factor of 1.8.

By using the GAP9's multi-core processor or the neural engine, we gain flexibility, as it is a multi-purpose architecture that allows to deploy of arbitrary networks while the MAX78000 is limited to a specific set of layers and only internal memory. In section Section IV-A we showed that we can gain accuracy and train for more classes if we can use bigger networks that can be deployed on GAP9 but not on the MAX78000.

b: COMPARISON TY:10-3-112:

We also deployed a more general-purpose TinyissimoYOLO network on GAP9, which has a higher input resolution and

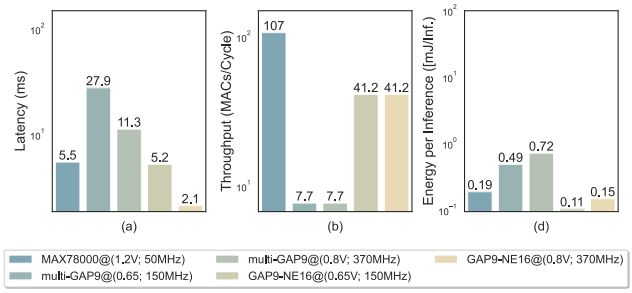

FIGURE 18. TY:3-3-88 performance comparison when deployed quantized to 8-bit on the CNN accelerated MAX78000 (50 MHz) MCU compared to the most energy efficient (150 MHz), the least latency (370 MHz) operating points of GAP9 and the neural engine of GAP9.

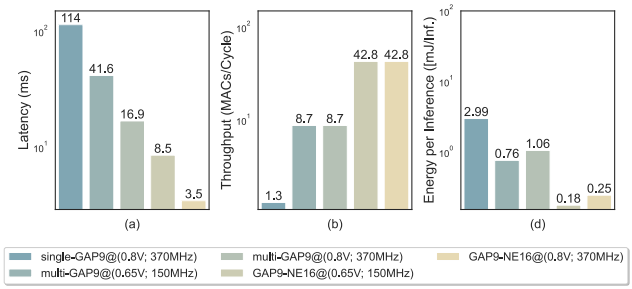

FIGURE 19. Performance comparison when deploying the network TY:10-3-112 quantized to 8-bit on the GAP9 running on the single-core (370 MHz), multi-core (150 MHz, 370 MHz) and on the neural engine of GAP9 (370 MHz).

10 detection classes. In Table 2 we note the network has 700k parameters. Furthermore, the input of 112 × 112 pixels RGB image consumes another 100 kB of memory while the in-between network calculations need at most approximately 375 kB of memory. As such, a microcontroller that needs to run such a network requires at least 1 MB of Flash while having 512 kB of RAM. Furthermore, this network clearly can not fit the MAX78000 anymore. We, therefore, deployed the network on GAP9 only. In particular, we deployed the network on one single core, on all nine cluster cores, and on the neural engine itself. Fig. 19 shows the comparison of the performances achieved. We note, despite the sheer size of the network, it runs within 3.5 ms on the neural engine, while being executed 32x and 3x slower on the single-core and multi-core implementation, respectively.

V. CONCLUSION

This work provides a comprehensive evaluation of various network adjustments for TinyissimoYOLO for edge processors with a 100s-of-KiB memory budget and in a 10s-of-milliwatt-range power envelope. We demonstrate the versatility of TinyissimoYOLO by training the network to detect up to 20 classes. Despite its small size, TinyissimoYOLO achieves remarkably high detection accuracy, coming close to the performance of YOLOv1 when trained on the entire PascalVOC dataset.

Additionally, we present an exhaustive investigation into the network's deployability with a fair benchmark and discussion of single-core microcontrollers and the benefit of parallelization. Furthermore, the novel RISC-V-based multi-core GAP9 processor is compared with the MAX78000 accelerator and the GAP9's neural engine (NE16). On NE16, we find that inference at the maximum clock frequency of 370 MHz only takes 2.12 ms, with an energy consumption per inference of 105 µJ, nearly half the energy consumption compared to the MAX78000 platform for a 3-class network with 88 × 88 input resolution. This network's architecture and size are largely dictated by the MAX78000 accelerator's limitations, which do not apply to GAP9's heterogeneous architecture - layers not supported by NE16 can still be efficiently mapped to the multi-core RISC-V cluster. We conclude that multi-core, general-purpose platforms are essential to achieving acceptable performance and efficiency levels. Heterogeneous systems incorporating domain-specific accelerators provide an efficiency boost in accelerated applications, but the presence of tightly coupled general-purpose processors is essential to maintain flexibility.

This is illustrated by the deployment of a larger, more powerful 10-class TinyissimoYOLO network using a larger input resolution of 112 × 112 pixels to GAP9's cluster and to NE16. Even with this more powerful network, GAP9 can perform 285 inferences per second, with the object detection update rate ultimately restricted by the exposure time of low-power cameras rather than inference latency. The remaining time between frames could be used for additional processing of the inference results, e.g. for object tracking. In conclusion, the energy efficiency and real-time capabilities of TinyissimoYOLO make it well-suited for low-power processors and applications such as always-on smart cameras, where it can perform object detection efficiently.

ACKNOWLEDGMENT

To Marco Fariselli, for his help in deploying TinyissimoYOLO on the neural engine of GAP9, to Thorir Mar Ingolfsson, for his help with DORY on GAP9, and to Jakub Mandula for proofreading this article.

(Julian Moosmann, Hanna Müller, and Nicky Zimmerman contributed equally to this work.)

REFERENCES

[1] K. Wei, L. Zhang, Y. Guo, and X. Jiang, "Health monitoring based on Internet of Medical things: Architecture, enabling technologies, and applications," *IEEE Access*, vol. 8, pp. 27468–27478, 2020, doi: 10.1109/ACCESS.2020.2971654.

[2] M. S. Diab and E. Rodriguez-Villegas, "Embedded machine learning using microcontrollers in wearable and ambulatory systems for health and care applications: A review," *IEEE Access*, vol. 10, pp. 98450–98474, 2022, doi: 10.1109/ACCESS.2022.3206782.

[3] Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and Industry 4.0: 360 degree comparison," *IEEE Access*, vol. 6, pp. 3585–3593, 2018, doi: 10.1109/ACCESS.2018.2793265.

[4] (2019). *Number of Internet of Things (IoT) Connected Devices Worldwide From 2019 to 2023, With Forecasts From 2022 to 2030 (in Billions) [Graph]*. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[5] N. Ling, X. Huang, Z. Zhao, N. Guan, Z. Yan, and G. Xing, "BlastNet: Exploiting duo-blocks for cross-processor real-time DNN inference," in *Proc. 20th ACM Conf. Embedded Networked Sensor Syst.*, New York, NY, USA, Nov. 2022, pp. 91–105, doi: 10.1145/3560905.3568520.

[6] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Computer Vision—ECCV 2020*. Cham, Switzerland: Springer, Aug. 2020, pp. 213–229.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Computer Vision—ECCV*, vol. 9905. Cham, Switzerland: Springer, Sep. 2016, pp. 21–37, doi: 10.1007/978-3-319-46448-0.

[8] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019, doi: 10.1109/ACCESS.2019.2939201.

[9] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li, X. Wang, and Y. Qiao, "InternImage: Exploring large-scale vision foundation models with deformable convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 14408–14419, doi: 10.1109/CVPR52729.2023.01385.

[10] R. Dagli and A. M. Shaikh, "CPPE-5: Medical personal protective equipment dataset," *Social Netw. Comput. Sci.*, vol. 4, no. 3, p. 263, Mar. 2023, doi: 10.1007/s42979-023-01748-7.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[12] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6517–6525, doi: 10.1109/CVPR.2017.690.

[13] J. Redmon and A. Farhadi, "YOLOV3: An incremental improvement," 2018, *arXiv:1804.02767*.

[14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOV4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[15] G. Jocher. (2020). *YOLOV5 By Ultralytics*. [Online]. Available: https://github.com/ultralytics/yolov5

[16] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei, "YOLOv6: A single-stage object detection framework for industrial applications," 2022, *arXiv:2209.02976*.

[17] C.-Y. Wang, A. Bochkovskiy, and H.-Y. Mark Liao, "YOLOV7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 7464–7475, doi: 10.1109/CVPR52729.2023.00721.

[18] G. Jocher, A. Chaurasia, and J. Qiu. (2023). *YOLO By Ultralytics*. [Online]. Available: https://github.com/ultralytics/ultralytics

[19] S. S. Saha, S. S. Sandha, and M. Srivastava, "Machine learning for microcontroller-class hardware: A review," *IEEE Sensors J.*, vol. 22, no. 22, pp. 21362–21390, Nov. 2022, doi: 10.1109/JSEN.2022.3210773.

[20] P. Warden and D. Situnayake, *TinyML: Machine Learning With Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers*. Newton, MA, USA: O'Reilly Media, 2019.

[21] Microsoft. (2021). *Neural Network Intelligence*. [Online]. Available: https://github.com/microsoft/nni

[22] J. Moosmann, M. Giordano, C. Vogt, and M. Magno, "TinyissimoYOLO: A quantized, low-memory footprint, TinyML object detection network for low power microcontrollers," in *Proc. IEEE 5th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2023, pp. 1–5, doi: 10.1109/AICAS57966.2023.10168657.

[23] D. Palossi, N. Zimmerman, A. Burrello, F. Conti, H. Müller, L. M. Gambardella, L. Benini, A. Giusti, and J. Guzzi, "Fully onboard AI-powered human-drone pose estimation on ultralow-power autonomous flying nano-UAVs," *IEEE Internet Things J.*, vol. 9, no. 3, pp. 1913–1929, Feb. 2022, doi: 10.1109/JIOT.2021.3091643.

[24] L. Lamberti, M. Rusci, M. Fariselli, F. Paci, and L. Benini, "Low-power license plate detection and recognition on a RISC-V multi-core MCU-based vision system," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Benin, May 2021, pp. 1–5, doi: 10.1109/ISCAS51556.2021.9401730.

[25] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988, doi: 10.1109/ICCV.2017.322.

[26] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440, doi: 10.1109/CVPR.2015.7298965.

[27] N. Zimmerman, T. Guadagnino, X. Chen, J. Behley, and C. Stachniss, "Long-term localization using semantic cues in floor plan maps," *IEEE Robot. Autom. Lett.*, vol. 8, no. 1, pp. 176–183, Jan. 2023, doi: 10.1109/LRA.2022.3223556.

[28] N. Zimmerman, M. Sodano, E. Marks, J. Behley, and C. Stachniss, "Constructing metric-semantic maps using floor plan priors for long-term indoor localization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2023, pp. 1366–1372, doi: 10.1109/IROS55552.2023.10341595.

[29] J. Mccormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger, "Fusion++: Volumetric object-level SLAM," in *Proc. Int. Conf. 3D Vis. (3DV)*, Sep. 2018, pp. 32–41, doi: 10.1109/3DV.2018.00015.

[30] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: An open-source library for real-time metric-semantic localization and mapping," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 1689–1696, doi: 10.1109/ICRA40945.2020.9196885.

[31] S. Rezwan and W. Choi, "Artificial intelligence approaches for UAV navigation: Recent advances and future challenges," *IEEE Access*, vol. 10, pp. 26320–26339, 2022, doi: 10.1109/ACCESS.2022.3157626.

[32] J. Crespo, J. C. Castillo, O. M. Mozos, and R. Barber, "Semantic information for robot navigation: A survey," *Appl. Sci.*, vol. 10, no. 2, p. 497, Jan. 2020, doi: 10.3390/app10020497.

[33] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of YOLO algorithm developments," *Proc. Comput. Sci.*, vol. 199, pp. 1066–1073, 2022.

[34] D. Seichter, M. Köhler, B. Lewandowski, T. Wengefeld, and H.-M. Gross, "Efficient RGB-D semantic segmentation for indoor scene analysis," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 13525–13531, doi: 10.1109/ICRA48506.2021.9561675.

[35] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, Jan. 2015, doi: 10.1007/s11263-014-0733-5.

[36] A. Canepa, E. Ragusa, R. Zunino, and P. Gastaldo, "Detection-based video surveillance using deep neural networks on STM32 microcontroller," in *Proc. 29th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Oct. 2022, pp. 1–4, doi: 10.1109/ICECS202256217.2022.9970956.

[37] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–19.

[38] M. Risso, A. Burrello, F. Conti, L. Lamberti, Y. Chen, L. Benini, E. Macii, M. Poncino, and D. J. Pagliari, "Lightweight neural architecture search for temporal convolutional networks at the edge," *IEEE Trans. Comput.*, vol. 72, no. 3, pp. 744–758, Mar. 2023, doi: 10.1109/TC.2022.3177955.

[39] E. Liberis, L. Dudziak, and N. D. Lane, "UNAS: Constrained neural architecture search for microcontrollers," in *Proc. 1st Workshop Mach. Learn. Syst.*, New York, NY, USA, 2021, pp. 70–79.

[40] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "Hardware-aware neural architecture search: Survey and taxonomy," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, Aug. 2021, pp. 4322–4329.

[41] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[42] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021.

[43] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, P. Warden, and R. Rhodes, "TensorFlow lite micro: Embedded machine learning for TinyML systems," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 800–811, Aug. 2021.

[44] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2 KB RAM for the Internet of Things," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1935–1944.

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

[46] L. Wulfert, C. Wiede, M. H. Verbunt, P. Gembaczka, and A. Grabmaier, "Human detection with a feedforward neural network for small microcontrollers," in *Proc. 7th Int. Conf. Frontiers Signal Process. (ICFSP)*, Sep. 2022, pp. 14–22, doi: 10.1109/ICFSP55781.2022.9924667.

[47] 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, doi: 10.1109/AICAS54282.2022.9870017.

[48] 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.*, New York, NY, USA, Nov. 2022, pp. 934–940, doi: 10.1145/3560905.3568300.
[49] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, 2020, pp. 11711–11722.
[50] J. Lin, W. Chen, H. Cai, C. Gan, and S. Han, "Memory-efficient patch-based inference for tiny deep learning," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, 2021, pp. 2346–2358.
[51] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," 2021, *arXiv:2110.15352*.
[52] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 326–335, May 2020.
[53] G. Rutishauser, F. Conti, and L. Benini, "Free bits: Latency optimization of mixed-precision quantized neural networks on the edge," in *Proc. IEEE 5th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2023, pp. 1–5.
[54] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," 2019, *arXiv:1902.08153*.
[55] M. Spallanzani, G. Rutishauser, M. Scherer, A. Burrello, F. Conti, and L. Benini, "QuantLab: A modular framework for training and deploying mixed-precision NNs," in *TinyML Summit*. Burlingame, CA, USA: Hyatt Regency San Francisco, Mar. 2022. [Online]. Available: https://cms.tinyml.org/wp-content/uploads/talks2022/Spallanzani-Matteo-Hardware.pdf
[56] S. Jain, A. Gural, M. Wu, and C. Dick, "Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 112–128, Jun. 2020.
[57] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021, *arXiv:2103.13630*.
[58] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. W. Mahoney, and K. Keutzer, "HAWQV3: Dyadic neural network quantization," in *Proc. 38th Int. Conf. Mach. Learn.*, vol. 139, 2021, pp. 11875–11886.
[59] 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, doi: 10.1109/JSSC.2021.3114881.
[60] F. Conti, G. Paulin, A. Garofalo, D. Rossi, A. Di Mauro, G. Rutishauser, G. Ottavi, M. Eggiman, H. Okuhara, and L. Benini, "Marsellus: A heterogeneous RISC-V AI-IoT end-node SoC with 2–8 B DNN acceleration and 30%-boost adaptive body biasing," *IEEE J. Solid-State Circuits*, vol. 59, no. 1, pp. 128–142, Jan. 2024, doi: 10.1109/JSSC.2023.3318301.
[61] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "DORY: Automatic end-to-end deployment of real-world DNNs on low-cost IoT MCUs," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1253–1268, Aug. 2021, doi: 10.1109/TC.2021.3066883.

JULIAN MOOSMANN (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering and information technologies from ETH Zürich, Zürich, Switzerland, in 2019 and 2023, respectively, where he is currently pursuing the Doctor of Science degree with the Integrated Systems Laboratory in conjunction, Center for Project-Based Learning D-ITET.

From 2022 to 2023, he was a Research Assistant with the Center for Project-Based Learning D-ITET, ETH Zürich. His research interests include combination of computer vision, event-based sensing, low-power systems, wireless sensor networks, tiny machine learning/onboard intelligence, and battery-operated distributed systems.

HANNA MÜLLER (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering and information technologies from ETH Zürich, Zürich, Switzerland, in 2017 and 2020, respectively, where she is currently pursuing the Ph.D. degree with the Integrated Systems Laboratory. Her research interests include low-power systems, wireless sensor networks, and onboard intelligence—especially for obstacle avoidance and localization of nano-drones.

NICKY ZIMMERMAN (Graduate Student Member, IEEE) received the M.Sc. degree in informatics from Università della Svizzera Italiana, where she is currently pursuing the Ph.D. degree with IDSIA. Her M.Sc. thesis titled "Embedded Implementation of Vision-Based Navigation for Nano-Drones." Previously, she was with General Motors and Intel.

GEORG RUTISHAUSER (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering and information technology from ETH Zürich, in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree with the Integrated Systems Laboratory. His research interests include algorithms and hardware for reduced-precision deep learning and their application in computer vision and embedded systems.

LUCA BENINI (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1997.

He was the Chief Architect of the Platform2012/STHORM Project with STMicroelectronics, Grenoble, France, from 2009 to 2013. Currently, he holds the Chair of Digital Circuits and Systems with ETH Zürich, Zürich, Switzerland, and a Full Professor with the University of Bologna, Bologna, Italy. He has published more than 1000 peer-reviewed articles and five books. His current research interest includes energy-efficient computing systems' design from embedded to high performance.

Dr. Benini is a fellow of the ACM and a member of the Academia Europaea. He was a recipient of the 2016 IEEE CAS Mac Van Valkenburg Award and the 2019 IEEE TCAD Donald O. Pederson Best Article Award.

MICHELE MAGNO (Senior Member, IEEE) received the master's and Ph.D. degrees in electronic engineering from the University of Bologna, Bologna, Italy, in 2004 and 2010, respectively.

Currently, he is a "Privatdozent" with ETH Zürich, Zürich, Switzerland, where he is the Head of the Project-Based Learning Center. He has collaborated with several universities and research centers, such as Mid University Sweden, where he is also a Guest Full Professor. He has published more than 150 articles in international journals and conferences, in which he got multiple best paper and best poster awards. His research interests include wireless sensor networks, wearable devices, machine learning at the edge, energy harvesting, power management techniques, and extended lifetime of battery-operated devices.