ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA, TELECOMUNICAZIONI E TECNOLOGIE DELL'INFORMAZIONE

Ciclo 38

Settore Concorsuale: 09/E3 - ELETTRONICA

Settore Scientifico Disciplinare: ING-INF/01 - ELETTRONICA

EFFICIENT NEURAL NETWORK INFERENCE ON HETEROGENEOUS MCU PLATFORMS

Presentata da: Luca Bompani

Coordinatore Dottorato

Davide Dardari

Supervisore

Francesco Conti

Co-supervisore

Luca Benini

Daniele Palossi

Manuele Rusci

Esame finale anno 2026

Acknowledgements

First of all, I would like to thank Daniele Palossi and Manuele Rusci for their continuous guidance and discussions. Daniele, specifically, I would like to thank you for the fruitful discussions and for the never-ending painting tips, Manuele, for the unbounded patience with my depressing self, and the never-ending corrections to my papers. To both of you, I owe a debt of gratitude for being there and helping me grow as a person and as a researcher

A very special thanks goes to Francesco Conti, my actual advisor, for being present in all the ways that matter: from the technical deep-dives to the non-technical conversations that somehow still improved the work (or at least improved my mood), and of course for the legendary sproloqui at meal time which henceforth shall be known as "high-bandwidth interdisciplinary research in disguise."

I also want to thank Giuseppe, the fourth and last of my direct supervisors, for the late nights in the lab, the never-ending jokes, some of questionable humor at best, but that made us laugh anyway, and for the unending well of programming knowledge that you have provided, along with your contagious good humour. Thank you for the long discussions on anime, C, C++, gardening, and animal handling, a combination that, on paper, should not work, and yet somehow perfectly represents the spirit of this Ph.D, and above all, thank you for organizing every year the Neurosalama event.

My thanks also go to my advisor at NTNU, Stefano Cherubin, to whom I owe having allowed me to delve into the realm of compilers, weekly pubs visits, and terrible puns (which , actually, were not a new topic for me)

To the overarching "Benevolent Dictator" (also known as Luca Benini): thank you for keeping the whole machine well-fed and operational (insert pun on aritmetic intensity here). Thank you for your wisdom that arrived exactly when needed in the form of a clean, elegant comment to the paper that makes everything snap into place.

To my colleagues: thank you all for making the lab a place where work is intense, collaboration is real, and humor is a survival skill. In particular, I want to thank Lorenzo Lamberti for the help with the drones and their... let's call it "strong personality" and "creative interpretation of instructions." And thank you to Davide Nadalini for the long discussions and the many ideas. Thank you, Alberto Dequino, for our continuous meme exchange and mood kindredness.

A generic (but very sincere) thanks also goes to all the colleagues I have not mentioned: you are too many to list without turning this section into a phone book, but you absolutely contributed to these years more than you might realize.

Finally, I want to thank my extended friends for being there outside the lab bubble:

Pietro Grenzi, Luca Baraldi (ARALDI), Chiara Rossini, Eliana Delle Chiaie, Giulia Pezzuoli, Lorenzo Caretti (I'm thinking about increasing the number of r and t), Federico Bernardi (Chicco).

And above all, thank you to my mother and father, Patrizia Vincenzi and Claudio Bompani, for your constant support, patience, and encouragement. This journey may have been "mine" on paper, but it was never done alone.

If this thesis exists, it is because of all of you, your time, your ideas, your support, and your ability to tolerate me for three, and more, years straight. Which, again, should probably come with a certificate.

Abstract

This thesis presents a hardware–software co-design methodology for deploying deep neural networks (DNNs) on heterogeneous microcontroller (MCU) systems targeting extreme-edge applications. It addresses the growing need for near-sensor intelligence in domains such as autonomous robotics, wearable devices, and precision agriculture, where power, memory, and compute resources are tightly constrained.

Object detection serves as the main case study throughout this thesis, as it combines classification and regression tasks and thus provides a representative benchmark for evaluating both discrete and continuous prediction capabilities on ultra-low-power MCUs.

This study begins by analyzing the scaling behavior of neural networks on resource-limited platforms, quantifying the trade-offs between model complexity, latency, and accuracy. This analysis reveals the inherent limitations of multicore MCUs. I perform this evaluation on a nano-drone equipped with the GAP8 MCU, demonstrating that while scaling down models (e.g., MobileNetV2-SSD) improves latency, it significantly impacts accuracy, causing a reduction of 13% of mean average precision, making aggressive compression a limiting factor when multiple workloads must run on the same MCU.

Motivated by these findings, the thesis advocates the adoption of heterogeneous MCU architectures, combining general-purpose RISC-V cores with domain-specific accelerators to improve throughput and energy efficiency. I investigate this introduction using the GAP9 SoC as a representative heterogeneous MCU for real-world applications, exemplified by an automated pest detection node for precision agriculture. Leveraging its neural accelerator, the platform achieves efficient on-device inference, achieving 7.2× higher throughput and requiring 34× less energy than the GAP8 architecture performing the same workload, MobileNetV3-SSDLite.

Finally, the proposed co-design methodology is applied to two workloads, video object detection (VOD) and speech enhancement (SE), to demonstrate its generality. For VOD, a multi-resolution inference policy (MR2-ByteTrack) interleaves

low- and high-resolution frames, improving accuracy by 2.16% and reducing energy consumption by 43.6% on GAP9 w.r.t. a baseline object detector working only with high-resolution frames. For SE, a neural beamforming pipeline exploits mixed-precision computation: an `int8` neural network generates spectral masks on the accelerator, while a `float32` MVDR beamformer ensures numerical stability. The system achieves real-time operation (20 ms latency) at an energy cost of 17.28 mJ per inference.

Overall, this thesis demonstrates that hardware–software co-design, applied systematically across algorithmic and architectural dimensions, enables complex AI workloads to execute efficiently on ultra-low-power heterogeneous MCUs, bridging the gap between embedded constraints and modern deep learning requirements.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The adoption of Artificial Intelligence (AI), and in particular deep neural networks (DNNs), has been pervasive across scientific and industrial domains, bringing a profound shift from handcrafted statistical and rule-based systems to end-to-end learning paradigms [4]. This transformation is evident across diverse application areas. In computer vision, pipelines based on handcrafted descriptors such as SIFT and HOG [5, 6] have been replaced by convolutional neural networks that automatically learn invariant, data-driven visual features, achieving unprecedented accuracy in classification and detection tasks [7]. Similarly, in audio and speech processing, deep architectures have supplanted traditional signal-processing front ends based on Gaussian mixture models, learning complex time–frequency representations directly from data and reaching near human-level performance in recognition and synthesis [8, 9]. Beyond the applications presented in this thesis, comparable paradigm shifts have occurred in natural language processing, where manually engineered features, such as n-gram statistics, syntactic parsers, and latent topic models [10] have been replaced by neural architectures capable of learning hierarchical representations directly from raw text, dramatically improving translation fluency and contextual understanding [11]. Across these fields, and more, the transition from expert-crafted features to learned representations has yielded more expressive and adaptable models, at the cost of vastly increased computational and memory demands. Over the past decade, the scale of computation required for training and using state-of-the-art models has grown by more than six orders of magnitude, with parameter counts expanding from millions to hundreds of billions [12]. This exponential growth has driven a shift toward massive distributed training on General Purpose Graphical Processing Unit (GPGPU) and Tensor Processing Unit (TPU) clusters. Today, achieving state-of-the-art performance demands infrastructure capable of sustaining petaflop-scale compute throughput and terabytes of active model memory, often orchestrated across thousands of nodes [13]. However, many

emerging use cases, such as autonomous robotics [14], augmented reality [15], and wearable biomedical devices [16], require computation close to the data-generating sensors. Operating near the sensing front-end enables rapid feedback and real-time decision-making while, at the same time, preserving privacy by avoiding data transmission to the cloud. This proximity imposes stringent constraints on available power and physical size, severely limiting the available computational and memory resources. While novel platforms have been proposed for executing deep learning models at the edge, such as the NVIDIA Jetson family [17], these systems integrate powerful GPU-based accelerators and multi-core Central Processing Units (CPUs). Despite their impressive performance, such architectures are designed for relatively large autonomous platforms and remain unsuitable for wearable devices or nano-scale systems, such as nano-drones. They are still far too large and energy-demanding to be deployed in these highly constrained contexts. To enable intelligence at this extreme edge, I must instead rely on a smaller class of devices,e.g., MicroController Units (MCUs), that rely on parallel computing to achieve high compute throughput while satisfying both spatial and energy constraints. This constraint, however, reduces the available computational budget to only a few Giga-Operations Per Second (GOPS) and a few megabytes of on-chip memory, thereby motivating the development of more efficient neural network architectures, optimized deployment strategies, and specialized hardware accelerators capable of bridging the computational and performance divide with the SoTA of desktop-class devices. While previous works have already demonstrated that low-power MCUs can execute compact neural network workloads [18] and even deploy more complex architectures on constrained platforms [19], little attention has been given to the joint optimization of algorithms and hardware for such limited computational resources. In particular, the interplay between computational complexity, energy consumption, and model accuracy has rarely been analyzed in conjunction with the architectural characteristics of the underlying hardware. This thesis contributes a comprehensive hardware–software co-design methodology for executing deep neural network workloads on heterogeneous ultra-low-power microcontroller systems targeting extreme-edge applications. I consider as my main case study the object detection application, as it combines classification and regression tasks, thereby stressing both discrete and continuous prediction capabilities under constrained execution. This makes it an effective benchmark for assessing the behavior and efficiency of different neural network architectures on ultra-low-power MCUs. I begin by analyzing the scaling behavior of deep neural networks on resource-constrained platforms through a robotic case study on object detection for nano-drones. Three SSD-MobileNetV2 variants ($\alpha = 0.5,0.75,1.0$) are deployed on a parallel MCU, the GAP8, spanning 193,M–534,M MACs. This setup enables a controlled study of how compute demand translates into frame rate, power, and detection performance under a fixed embedded budget. Despite its higher complexity, the largest configuration ($\alpha = 1.0$) achieves the best operating point in this scenario, reaching

a mean detection rate of 90 % at 1.6 frame/s and 134 mW. Conversely, scaling the network down improves throughput but leads to a clear accuracy penalty (up to −13 % mAP), highlighting that aggressive compression is not "free" even when it enables faster inference.

These results confirm that increasing model capacity can remain beneficial even at the extreme edge, but they also expose a fundamental limitation of homogeneous multicore MCUs: the available throughput is often sufficient for a single perception pipeline, yet quickly becomes inadequate once autonomy requires multiple vision workloads to run concurrently. In practice, adding a second network—for instance an obstacle-avoidance model such as PULP-DroNet, which requires $\geq$ 8 frame/s to maintain an 80 % collision-avoidance probability—forces a strict reallocation of compute. Under a fixed power and throughput envelope, this typically means either time-multiplexing the workloads (reducing update rate) or further shrinking the networks to fit, both of which directly trade away performance: lower frame rates degrade responsiveness, while additional model scaling amplifies the accuracy losses already observed in the single-network setting.

This constraint motivates the transition to heterogeneous MCU architectures combining general-purpose cores with domain-specific accelerators. In my pest-detection study, I show the higher efficiency of heterogeneous MCUs, exemplified by the GAP9, which achieves, running a Mobilenetv3SSDLide network, 10.9 MAC/cycle, 7.2× over GAP8, with 9.5× faster inference, requiring 147 ms and 34 × less energy, 4.73 mJ. This improvement enables an estimated lifetime of ∼199 days with a duty-cycled deployment powered by a 1000 mA h battery at 3.7 V, demonstrating that heterogeneous MCU designs can sustain high-performance deep inference while preserving the ultra-low-power operation required for long-lived edge intelligence. Building upon this foundation, the thesis presents application-driven evaluations of the proposed co-design methodology. The first case study revisits a Video Object Detections (VODs) pipeline, where interleaving two down-scaled frames for every high-resolution frame within the MR2-ByteTrack framework results in an average accuracy improvement of 2.16% and reduces energy consumption of 43.6% on the GAP9 MCU compared to a baseline high-resolution only inference scheme. Finally, to demonstrate the generality of the approach beyond vision tasks, the methodology is applied to multi-microphone speech enhancement (SE). This is obtained through neural beamforming, which performs spatial filtering by combining signals from multiple microphones to enhance sound from a target direction while attenuating background noise. The implemented pipeline achieves real-time operation with a 20 ms ms inference latency and an energy cost of 17.28 mJ per inference, confirming the effectiveness and versatility of the proposed co-design framework across diverse sensing modalities.

1.1 Thesis outline and contributions

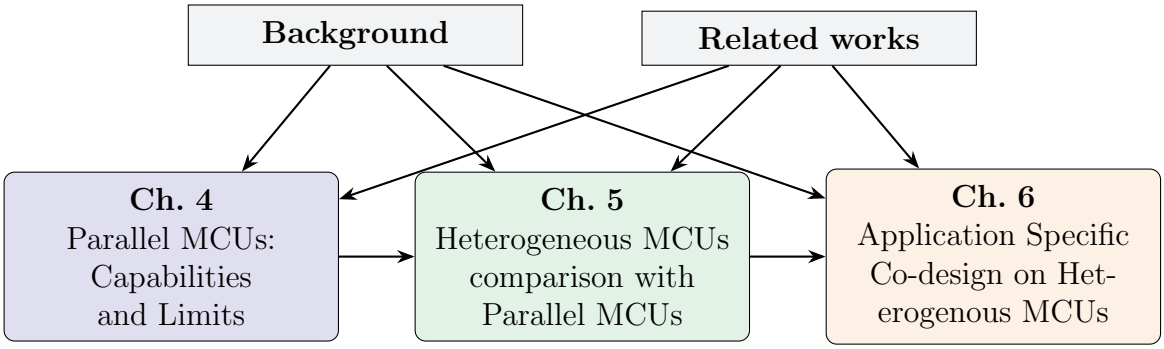

Figure 1.1. Overview of the thesis structure and experimental progression split over its three main chapters.

An overview of the structure of this thesis is depicted in Figure 1.1. The content of each chapter is summarized in the following.

Chapter 2 - Related Work This chapter surveys the state of the art underpinning this thesis. It first reviews the evolution of object detection, highlighting the emergence of lightweight architectures specifically designed for embedded inference. The analysis then considers their prospective deployment on nano-Unmanned aerial vehicless (UAVs), complemented by an examination of navigation strategies and CNN-based visual obstacle avoidance for this class of devices. Together, these studies provide the algorithmic and control background relevant to autonomous exploration in unknown environments, where object detection must operate as part of an integrated perception–control pipeline. The discussion subsequently addresses the specialization of object detection for agricultural pest monitoring, tracing the shift of this application from cloud-based processing to edge-deployed systems. It then extends to video object detection, viewed as a generalization of single-frame object detectors. Finally, to broaden the analysis to a distinct sensing modality, the chapter introduces beamforming for multichannel speech enhancement as an audio counterpart, illustrating how analogous computational, memory, and energy constraints arise. Throughout, the analysis emphasizes the key performance metrics, accuracy, latency, memory footprint, and energy efficiency,that motivate the transition toward heterogeneous embedded architectures for intelligent processing at the extreme edge.

Chapter 3 - Background This chapter introduces the GAP8 and GAP9 SoCs from GreenWaves Technologies, two generations of ultra-low-power multicore microcontrollers designed for edge-AI applications. The GAP8 provides a parallel RISC-V architecture for general-purpose compute, while the GAP9 extends it with

dedicated hardware accelerators for deep learning. The chapter also outlines the AutoTiler deployment framework, which automates data tiling and memory management to efficiently map neural networks onto these platforms.

Chapter 4 - Object Detection on a Nano-Drone Using a RISC-V Multi-Core MCU In this chapter, I investigate the feasibility and practical limitations of executing deep neural networks on resource- and energy-constrained parallel microcontroller units (MCUs). Specifically, I focus on the GAP8 platform, a parallel ultra-low-power processor representative of multicore MCUs. The analysis builds upon two experimental studies that collectively explore the trade-offs between computational efficiency, model complexity, and task performance in embedded artificial intelligence.

I begin by examining the deployment of a CNN for object detection, based on the MobileNetV2-SSD architecture. The MobileNetV2 model introduces a width multiplier parameter, commonly denoted as α, which uniformly scales the number of channels in each convolutional layer. This parameter allows balancing computational cost and model accuracy, effectively controlling the network's depth and representational capacity. By varying α across different configurations, $\alpha = 0.5,0.75,1.0$, I evaluate the trade-off between execution latency and detection accuracy when executing on the GAP8 MCU.

The analysis reveals that while lightweight configurations yield faster inference (4.3 frames/s for $\alpha = 0.5$ 2.7× faster than $\alpha = 1.0$, their accuracy degradation is non-negligible for perception tasks of practical relevance (13% loss in mAP reaching 37%). Ultimately, the full-width configuration ($\alpha = 1.0$) achieves the most favorable performance-to-accuracy balance under the given hardware constraints, achieving a detection rate of 90%, and it is therefore selected as the baseline for subsequent experiments. Building upon this finding, I integrate the object detection network with an additional model dedicated to obstacle avoidance, demonstrating that the concurrent execution of multiple perception tasks quickly saturates the available computational throughput. This observation underscores the necessity of employing more capable embedded hardware when high-throughput inference or multi-network integration is required, a consideration that becomes particularly critical in autonomous robotic systems where all computation must occur on-board.

Chapter 5 - Heterogeneous MCU for On-Device Pest Detection In the previous chapter, I analyzed the capabilities of modern multi-core MCUs, showing that these devices can sustain increasingly complex workloads, including convolutional neural networks, within tight memory and power budgets. Nevertheless, the evaluation also revealed the limitations that arise when executing compute-intensive tasks entirely on general-purpose cores, particularly in latency-critical applications.

To overcome these constraints, recent embedded platforms have adopted heterogeneous architectures that integrate domain-specific accelerators alongside conventional processing cores. Such systems, exemplified by the GAP9 System on Chip (SoC), combine multiple RISC-V cores with a dedicated convolutional engine, offering substantial gains in performance and energy efficiency compared to baseline MCUs such as the GAP8.

To evaluate these benefits, I developed an automated pest detection sensor for precision agriculture targeting the codling moth (*Cydia pomonella*). Two detection pipelines were implemented: a traditional Viola–Jones classifier and a CNN-based MobileNetV3-SSDLite. I test my algorithms on two datasets on the first, both algorithms achieved comparable accuracy (81.7 % vs. 83 %), while on a second set containing smaller insects, the CNN generalized better, reaching 72 % accuracy, 27 % higher than Viola–Jones.

From a computational standpoint, the heterogeneous GAP9 significantly outperforms the GAP8. The Viola–Jones algorithm runs in 221 ms per 320×240 frame, ×1.47 faster than on GAP8, thanks to a higher clock speed (+37 %) and an additional 64 kB of low-level memory. More strikingly, the MBNV3-SSD executes in 147 ms using the convolutional accelerator, ×9.5 faster than on GAP8, reaching 10.9 MAC/cycle efficiency, or ×7.2 higher than the previous generation.

These results show that heterogeneous MCUs can enable accurate CNN-based pest detection previously attainable only on processors consuming over ×15 more energy [20]. Moreover, with a duty-cycled operation powered by a 1000 mA h, 3.7 V battery and one image analyzed every 30 seconds, the system achieves an estimated lifetime of nearly 199 days, demonstrating how heterogeneous integration unlocks real-time intelligence under ultra-low-power constraints.

Chapter 6 - Application Specific Acceleration for Heterogeneous MCUs
In this final chapter, I apply the proposed hardware–software co-design methodology to two representative deep-learning workloads, VOD and SE, demonstrating how aligning algorithmic design with architectural features enables efficient execution of complex models on ultra-low-power platforms. The approach emphasizes adapting computation to available resources while exploiting application-specific structure, jointly optimizing precision, latency, and energy efficiency within the constraints of heterogeneous microcontroller-class systems.

For VOD, I target vision-based perception tasks in embedded systems such as miniature drones and surveillance nodes, where memory and compute budgets are limited to a few megabytes and tens of milliwatts. To address these constraints, I introduce a multi-resolution inference strategy that interleaves full- and low-resolution frames, reducing the per-frame cost by up to 43% without increasing model parameters. A Kalman-based tracker (ByteTrack) maintains object identity across frames, while

a probabilistic Rescore mechanism refines category predictions over time, mitigating transient misclassifications. Deployed on the GAP9 SoC, this MR2-ByteTrack pipeline achieves up to 1.76× lower latency compared to single-resolution inference and improves mean Average Precision by +2.16%, sustaining real-time throughput within a peak power envelope of only 72 mW.

For multi-channel SE, I design a neural beamforming pipeline for hearable devices, combining a lightweight `int8` convolutional mask estimator with a `float32` MVDR beamformer. This mixed-precision division of labor preserves numerical stability in spatial filtering while exploiting quantization tolerance in the neural stage. An interleaved execution strategy overlaps beamforming with weight estimation, satisfying the real-time constraint of 20 ms per frame. The system achieves a STOI score of 0.88 at an energy cost of 17.28 mJ per beamforming inference, corresponding to roughly 16 hours of continuous operation on a tiny 100 mAh 3.7 V battery.

Together, these two case studies demonstrate that co-optimizing algorithms and hardware can sustain high perceptual accuracy and real-time performance within milliwatt-level power budgets, advancing the feasibility of intelligent sensing at the extreme edge.

Chapter 2

Related Work

This chapter surveys the state of the art relevant to this thesis. I begin by reviewing object detection, which represents the main vision workload considered in this thesis, outlining its evolution from two-stage to one-stage detectors and the emergence of lightweight architectures tailored for embedded inference. I then examine its deployment on parallel multi-core MCUs in nano-UAV platforms, distinguishing the neural inference workload from the exploration strategy that governs navigation. Subsequently, I focus on agricultural pest detection, a domain-specific instance of object detection, emphasizing the transition from cloud-based to edge-based systems and the enabling role of heterogeneous MCUs with dedicated CNN accelerators. Finally, I extend the discussion to video object detection, viewed as the temporal evolution of single-frame object detection models, and to multichannel speech enhancement with beamforming, which, although distinct in modality, shares comparable computational, memory, and energy constraints. Across these topics, I highlight the central metrics of accuracy, latency, memory footprint, and energy efficiency that underpin the motivation for heterogeneous embedded architectures addressed in this thesis.

2.1 Object Detection

The advent of deep Deep Neural Networks (DNNs) has profoundly transformed object detection, the task of simultaneously localizing and classifying objects within an image. Early DNN-based detectors, such as R-CNN and its successors Fast R-CNN and Faster R-CNN [21], relied on a two-stage pipeline in which a first network generates a set of region proposals—candidate areas likely to contain objects—that are subsequently refined and classified in a second stage. Although this approach achieved high accuracy, it introduced data-dependent computational costs and large memory footprints, resulting in variable inference latency and rendering

such models unsuitable for real-time or embedded applications where determinism and resource constraints are critical.

To address these limitations, one-stage detectors were introduced, performing localization and classification in a single forward pass over the image. Representative examples include YOLO (You Only Look Once), SSD (Single Shot Detector) [22], and RetinaNet [23], which directly regress bounding-box coordinates and class probabilities without the intermediate region-proposal step. Their streamlined architectures—composed of a convolutional backbone followed by lightweight detection heads—achieve fully deterministic execution and lower latency, establishing one-stage detectors as the prevailing paradigm for time-critical applications.

Despite their improved efficiency, early one-stage models still exhibited footprints too large for ultra-constrained devices such as microcontrollers or nano-aerial robots. This limitation motivated the design of lightweight backbones optimized for embedded deployment. MobileNetV1 and V2 [24, 25] pioneered the use of depthwise separable convolutions, drastically reducing multiply–accumulate operations while preserving representational capacity. Combined with simplified detection heads, the resulting SSDLite architecture achieved an mAP of 22.1 % on COCO [26] with only 4.3 M parameters and 0.8 G MACs, demonstrating that real-time detection could operate within sub-watt power envelopes.

Building upon these advances, the EfficientNet and EfficientDet families [27] exploited neural-architecture search and compound scaling to jointly optimize accuracy, efficiency, and model size. In particular, EfficientDet-D0 employed a bidirectional feature-pyramid network (BiFPN) and a decoupled detection head, reaching an mAP of 34.6 % with only 2.5 G MACs and 3.9 M parameters. These design principles provided a systematic framework for balancing complexity and performance, paving the way for further miniaturization. Subsequent works such as NanoDet and YOLOX-Nano [3, 28] extended these ideas to the extreme, introducing sub-megabyte detection frameworks suited for low-power edge devices. For instance, NanoDet-Plus integrates adaptive label assignment and dynamic soft targets, achieving an mAP of 27 % on COCO with fewer than 0.5 G MACs, while YOLOX-Nano attains 25.8 % mAP with 0.9 M parameters and 0.36 G MACs. These models epitomize the culmination of a decade-long evolution—from the highly accurate but resource-hungry two-stage detectors to modern, ultra-efficient architectures that balance precision, throughput, and energy consumption—enabling real-world deployment in embedded and autonomous platforms.

Parallel to these CNN-based approaches, the introduction of Transformer-based detectors further diversified the design space. DETR [29] reformulated detection as a set-prediction problem optimized via bipartite matching, eliminating the need for handcrafted anchors but at the cost of high memory usage (159 MB) and slow convergence. Subsequent mobile-oriented Transformers, such as MobileViT and its

derivatives [30, 31, 32], achieved an mAP of 19.3 % with only 1.2 M parameters in the backbone, bridging CNN-Transformer efficiency gaps

2.2 AI-powered nano-drones.

Recently, several AI-based visual pipelines have been brought to nano-drone systems. *Lamberti et al.* [33] developed a CNN for end-to-end autonomous visual navigation that achieves an inference throughput of 160 FPS on a nano-drone, thanks to a reduced CNN complexity (down to 1.5 MMAC). This network design has two outputs: the steering angle and the collision probability. The latter is trained to detect any obstacle without providing information about its class or position within the image frame. Similarly, *NanoFlowNet* [34] is a CNN for dense optical flow that aims at the obstacle avoidance task. It outputs per-pixel information about the obstacles in the scene, yet it does not give any information about their class. *Palossi et al.* [35] deployed a CNN for human pose estimation aboard a nano-drone, allowing a drone to follow the movement of a human subject. However, this network is not designed to recognize multiple subjects in the same frame or different classes of subjects. In contrast, my work aims to deploy an object detector to a nano-drone to detect multiple instances and classes of objects in its field of view.

Computer vision-based navigation techniques, based on complex feature extraction pipelines [36] or simultaneous localization and mapping (SLAM) techniques [37], are reliable for autonomous robotic navigation. Still, as SLAM-based approaches have large memory requirements and rely on computationally intensive algorithms, they are exclusive to large UAV systems carrying heavy and high-power embedded computing systems [37].

The state-of-the-art autonomous exploration approaches for nano-drones, which suffer from limited computational power and memory, take advantage of bio-inspired (or bug-inspired) algorithms, which rely on lightweight state-machine-based algorithms and low-power sensor readings[38]. For example, [39] compares two bio-inspired exploration policies: the first changes the heading direction randomly after detecting an obstacle, while the second follows the obstacle's boundaries. Similar to the second one, [38] implements a policy that follows the walls of a room, called wall-following, while [40] describes a spiral motion. Moreover, there is a subcategory of bug-inspired algorithms that uses ranging measurements for navigating towards a target point [41, 42], even in the presence of obstacles. In my work, I adapt such bio-inpired ranging-based algorithms to my flying nano robotic platform and, for the first time, I integrate them with a visual object detection pipeline, evaluating the effectiveness of the exploration policy within a search mission.

Focusing on the porting of object detectors on embedded systems, *Tran Quang Khoi et al.* [43] deploy an SSD network (SSDLite-MobileNetV2) with a similar

architecture as the one used in this thesis onto a RaspberryPi B3+ mounted on a standard-sized drone. The model execution reaches 0.71 FPS, lower than my obtained throughput. More importantly, this solution exceeds both the power (up to 3 W) and size constraints of the nano-drone system considered in Chapter 4. *Lamberti et al.* [19] presented an SSD algorithm for license plates detection on a static multi-core MCU node (GAP8), achieving up to 1.6 FPS with a power consumption of 117 mW. Starting from this previous work, I design and integrate an SSD-based object detection CNN onto a nano-drone capable of exploring the environment while detecting specific objects.

2.3 Automated Pest Detection Systems

In Section 2.1, I introduced the general background on neural network based object detection algorithms, outlining their evolution. In this section, I specialize on their application for the monitoring of pests in agriculture, focusing on the different *systems* that integrate such algorithms into field-deployable smart traps or monitoring platforms.

Computer vision systems for codling moth detection can be broadly categorized into two classes: **cloud-based** and **edge-based** approaches. In cloud-based systems, images captured by in-field traps are transmitted to a central server for analysis, typically using CNN-based classifiers. For example, *Preti et al.* [44] combined a particle filter with a CNN leveraging morphological features of *Cydia pomonella*, while *Suarez et al.* [45] implemented a two-stage pipeline with a Canny edge detector followed by a CNN classifier, achieving 94.8% accuracy. Commercial solutions such as *TrapView*[1] similarly rely on server-side image analysis, typically processing only a few images per day.

Conversely, **edge-based systems** perform near-sensor processing directly within the trap to minimize data transmission and energy consumption. Several works employ high-end embedded boards: *Suto* [46] used a Raspberry Pi Zero W coupled with LoRa communication, combining Selective Search with MobileNetV2 for classification, requiring 195 s per inference. *Albanese et al.* [47] integrated a Raspberry Pi 3 with an Intel Neural Compute Stick to accelerate CNN inference, reaching 96% accuracy at 5 W of power. Similarly, *Vcirjak et al.* [48] ran EfficientDet on a Raspberry Pi 4, achieving 99.3% accuracy with 5 W peak consumption. While accurate, these designs rely on multi-watt processors, limiting large-scale deployment.

Efforts toward **ultra-low-power** solutions include the ESP32-based system of

[1] `https://trapview.com/`

Schrader et al. [49], which only captures and transmits images, and the OpenMV-based approach in [20], using an STM32H7 MCU running Selective Search and MobileNetV2 at approximately 500 mW. *Brunelli et al.* [2] proposed a GAP8-based trap employing a Gaussian Mixture Model for motion-based detection and a CNN classifier, consuming up to 52 J per transmitted image. A later work [50] introduced a lightweight single-stage Viola–Jones detector on GAP8, halving the execution time compared to the previous pipeline.

Building on these foundations, this thesis compares a traditional Viola–Jones detector and a CNN-based object detector on a **novel heterogeneous MCU** featuring a dedicated CNN accelerator. The results demonstrate that, for the first time in this application domain, a CNN-based detector can achieve comparable energy consumption to a handcrafted algorithm (4.85 mJ vs. 4.61 mJ), narrowing the efficiency gap between classical and deep-learning-based approaches for near-sensor pest monitoring in agricultural traps.

2.4 Video Object Detection

A first class of techniques for video object detection uses 3D convolutions instead of 2D ones [51]. This conceptually simple approach is not real-time and demands a large memory footprint to store high-dimensional activation tensors, exceeding the memory available on low-end MCU systems. An alternative line of research aggregates information across frames from region proposals obtained by two-stage detectors: in [52] using optical flow, or convolutional-based trackers in [53, 54], and transformer-based memory layers in [55]. However, the cost of region proposal aggregation exceeds the available budget for MCUs.

A more lightweight group of techniques builds on top of one-stage detectors, like the one presented in 2.1, by modeling the temporal correlation of detections. [56] uses dynamic programming to link objects across frames based on the Intersection over Union (IoU) metric. *Seq-Bbox* [57] and *Motion-based Seq-Bbox* [58] refine the outputs of the underlying object detector using the temporally associated detection information, marking a mAP of 80.9 and 72.7 , respectively, on the ImagenetVID dataset (+6.9 and +5.5 mAP points more than frame-by-frame inference). Conversely, my work introduces a novel probabilistic algorithm for aggregating scores over time, while these methods make the average and, differently from my solution, do not achieve any thorough improvement concerning the baseline.

More recently, transformer architectures have also been widely adopted for VOD tasks. The TransVOD Lite [59] uses Deformable DETR object detector [60] and scores a mAP50 of 83.7 on ImagenetVID with a SwinT transformer backbone [61]. The latter features a total of 45.9 M parameters and a computational cost of 8.89 GMAC per frame, exceeding the resources of an MCU system. In the best

configuration, TransVOD Lite achieves +3.5 mAP score vs. the object detector baseline by processing a batch of 15 frames. This setting requires 272 MB to store the intermediate features (15× memory increases compared to single frame execution). If applied on individual frames, the method, instead, reduces the mAP by 3.6 w.r.t. the Deformable DETR detector. On the contrary, my method (i) does not store multiple features across frames, but only the bounding boxes (9.5 kB) and (ii) leads to an mAP increases vs. the baseline when running frame-by-frame, i.e., it does not require a batch of images for the inference.

YOLOV [62] uses a transformer on top of a YOLOX detector to fuse the features extracted from multiple frames and refine the produced detection. This approach reaches a mAP of 85.5 on ImagenetVID. However, tuning the transformer requires a training stage, marking a significant difference w.r.t. my training-free method, which can be applied to any pre-trained model without additional computation.

2.4.1 Increasing VOD Throughput

A set of methods proposed data-dependent DNN execution pipelines to skip part of the inference operations based on the information carried by the input. The work of *Zhu et al.* [63] introduces an auxiliary but lightweight network, called dynamic-resolution network (DRNet), to determine the resolution of the input image before running the inference. The policy predicts a higher downsize ratio for images with "easy" objects to recognize. Concerning this thesis, I adopt a static resize policy avoiding the 4.9 M parameters overhead introduced by their DRNet, while achieving similar efficiency gains (∼40%).

On the contrary, other works speed up the processing by considering the temporal information and not only the content of a single frame. [64] avoids the computation of the features from specific regions of the current frame by copying the features computed in previous frames. They use an auxiliary lightweight ResNet-8 network to predict the accuracy gain of computing new features vs. copying the previous ones. Compared to this approach, I do not store the activations but only preserve the detected bounding box coordinates from the analysis of previous ones. In the case of the NanoDet-Plus network with a 320 × 320 px input image, this method would lead to a memory overhead of ∼50 MB vs. the 9.5 kB of my method. Furthermore, the ResNet-8 auxiliary network requires an extra training procedure using reinforcement learning to learn an efficient policy.

Liu et al. [65] couple a large model with a small one (4.4 M vs. 0.5 M parameters). Similar to my approach, they also interleave the two models' execution, and to recover from accuracy losses due to the smaller model, they placed an LSTM layer to connect the detection performed across time. A similar idea is also employed in [66], where they interleave different neural networks of varying complexity to determine the pose of a person in front of a drone. These methods deploy the

two models on the target platforms, increasing the memory cost compared to the baseline single-model inference. On the contrary, my framework does not require any extra parameters. It is applied to a pre-trained model, saving the cost of training a family of scalable models for the interleaved execution.

2.5 Multi-channel Speech Enhancement

Speech enhancement algorithms deployed on MCUs have predominantly relied on single-channel recurrent neural networks (RNNs). An early example is TinyLSTM [67], deployed on the STM32F746VE microcontroller, where the authors start from baseline LSTM-based SE architectures and progressively compress them through structured pruning of hidden units, skipping RNN state updates, and 8-bit quantization of weights and activations. Employing training-aware quantization to reduce the degradation introduced by quantization. Another representative solution is given by the NNoM framework [68], which demonstrates the deployment of the RNNoise [69] model on a single-core ARM Cortex-M MCU. Owing to its compact GRU layers, roughly 87 thousand parameters, and carefully constrained activation ranges, RNNoise can be effectively quantized to 8 bits with minimal loss in quality. Both of these approaches, however, are limited to relatively simple monocore MCUs. In contrast, more recent work has explored multi-core MCUs such as GAP9. For instance, Rusci *et al.* [70] introduced a mixed-precision deployment scheme that quantizes recurrent layer parameters and activations to `int8`, while retaining `float16` precision for other tensors. This strategy achieves an almost lossless deployment, with only a 0.007 degradation in STOI.

Despite these advances, all of the above methods remain restricted to a single-channel SE, and none address the more challenging problem of deploying multichannel, beamforming-based SE pipelines under MCU-class resource constraints. More recently, several neural architectures have been proposed for multichannel SE [71, 72, 73, 74]. These approaches span a broad design space, from UNet-like encoder–decoder structures [74] to Transformer-based models [71, 72, 73], and demonstrate strong denoising capabilities. However, despite their promising accuracy and, in some cases, computational requirements comparable to those of my pipeline [71], none of these methods have yet been deployed on real hardware platforms. Conversely, in this thesis, I present an end-to-end deployed solution. This brings multichannel SE with beamforming into an MCU-class device under realistic deployment constraints.

Finally, I also report the work of Wang*et al.*[75]. They do not use the MVDR beamformer for speech enhancement, but rather use it to estimate the position of a sound source. They report taking 1.5 seconds to run the MVDR algorithm (6.5× slower than my weight estimation step), falling short of real-time on a 200 MHz PIC32MZ microcontroller.

Chapter 3

Background

This chapter introduces the MCUs employed as computing substrates throughout this thesis, namely the *GAP8* and *GAP9* SoCs developed by GreenWaves Technologies. These systems represent two generations of parallel and heterogeneous MCUs designed for ultra-low-power edge computing, supporting a broad spectrum of applications such as embedded vision, robotics, and sensor fusion. While the subsequent chapters will detail specific deployment and optimization methodologies targeting these platforms, it is important to emphasize that the techniques and principles discussed are general and can be applied to a wide range of modern heterogeneous MCUs, sharing similar architectural traits such as multi-core processing capabilities, hierarchical memory organization, and specific hardware accelerators.

The chapter is structured as follows. Section 3.1 describes the architecture of the *GAP8* SoC, introducing its dual-domain organization and memory hierarchy. Section 3.2 presents its successor, the *GAP9*, which extends the same architectural principles while integrating specialized hardware accelerators for deep learning workloads. Section 3.3 compares the two systems, highlighting the main architectural advancements that make GAP9 a more capable and energy-efficient platform for modern embedded Artificial Intelligence (AI) applications. In addition to the architectural overview of the GAP platforms, this chapter introduces the software toolchains that enable neural network deployment on heterogeneous MCUs. These compilers bridge high-level models and optimized low-level code, managing tiling, memory allocation, and data transfers to fully exploit the computational capabilities of the GAP8 and GAP9 SoCs. Section 3.4 reviews the main deep learning compilers for edge devices, with emphasis on the Autotiler tool.

3.1 The GAP8 Architecture

The *GAP8* SoC is a parallel microcontroller platform that combines a low-power control core, the *Fabric Controller (FC)*, with a programmable parallel accelerator domain, the *Cluster (CL)*, see Fig. 3.1A for a simplified representation of the architecture. This dual-domain architecture enables the efficient execution of both control-oriented and data-intensive workloads. The FC domain handles system management, peripheral control, and lightweight tasks, while the CL is activated dynamically to execute highly parallel computational kernels, such as those found in Convolutional Neural Networks (CNNs) and Digital Signal Processings (DSPs).

The two domains communicate through a shared memory space, supported by a hierarchical interconnect and hardware synchronization primitives. This allows seamless offloading of compute-intensive functions from the FC to the CL without explicit context switching or message passing overhead.

The Fabric Controller is an advanced MCU built around a RISC-V core extended for energy-efficient digital signal processing. It includes an instruction cache, a fast on-chip data memory, and a complete set of peripherals such as Quad Serial Peripheral Interface (QSPI), Inter Integrated Circuit (I2C), Integrated Interchip Sound (I2S) interfaces, enabling the parallel acquisition of images, audio, and vibration data. Additionally, a four-channel Pulse-Width Modulation (PWM) controller supports direct actuation, e.g., for drone motor control or robotic servos.

Peripheral data transfers are managed by a multi-channel Direct Memory Access (DMA) engine, which minimizes the FC's workload by handling block-level data movement independently. The SoC provides up to 512 kB of on-chip L2 memory and a ROM storing the boot code, which also implements a secured boot mechanism based on eFUSE-stored keys. External memory expansion is supported through a hyperbus double data rate interface.

The GAP8 employs a unified address space, allowing all cores to access any on-chip memory. Access latency increases progressively across memory levels (L1, L2, and external L3), and a multi-channel DMA controller is provided to asynchronously transfer data between levels, thus hiding access latencies.

The cluster domain resides on a separate voltage and frequency island and is power-gated when not in use. It contains eight RISC-V cores identical to the one in the FC, which simplifies software portability across domains. The CL is designed for highly parallel workloads, supporting shared-memory programming models such as OpenMP.

The eight cores share a tightly coupled 64 kB L1 scratchpad memory, accessible in a single clock cycle with low contention (below 10% on data-intensive kernels). An optimized interconnect ensures high bandwidth between cores and memory banks,

while a shared program cache improves efficiency for data-parallel code.

A hardware synchronization unit (*HW Sync*) manages fine-grained task synchronization, barrier events, and thread scheduling. It also handles automatic clock gating, ensuring that idle cores enter a fully gated state to eliminate dynamic power consumption.

The GAP8 cores feature a four-stage in-order pipeline compliant with the RISC-V IMC subset. To improve throughput on DSP-centric workloads, several custom ISA extensions are implemented, including hardware loops, post-modified load/store instructions, single-cycle multiply–accumulate and complex arithmetic, rounding and normalization primitives, and SIMD operations on 8-bit and 16-bit vectors. These extensions make the architecture particularly well-suited for low-power signal and image processing.

3.2 The GAP9 Architecture

The *GAP9*, see Fig. 3.1B for a simplified representation of the architecture, is the second generation in the GreenWaves family of parallel MCUs. It maintains the dual-domain organization of its predecessor while improving energy efficiency, increasing total on-chip memory and clock speed. In addition to the RISC-V cores, it integrates a specialized convolution accelerator (*NE16*) and floating-point units (FPUs), significantly improving throughput on mixed-precision neural workloads.

The host domain of the GAP9 is similar to the GAP8 one, presenting as only difference a larger L2 memory, higher clock speed on the fabric controller, up to 370 MHz. The cluster domain of GAP9, Fig. 3.1B, comprises nine RISC-V cores (one more than GAP8) and a dedicated *NE16* accelerator. The NE16 consists of an array of $9 \times 9 \times 16$ 8×1-bit MAC units, capable of executing up to 150 8-bit MAC operations per clock cycle. This results in up to a $\sim$5× speed-up for convolutional layers compared to the CPU-only execution in GAP8.

Both the RISC-V cores and the NE16 share a tightly coupled 128 kB L1 scratchpad memory accessible within a single clock cycle. The cluster DMA enables asynchronous data movement between L1 and L2, ensuring high compute utilization. By decoupling memory transfer from computation, GAP9 achieves very high operational efficiency when executing tiled CNN kernels or signal processing tasks.

3.3 Architecture Comparison Between GAP8 and GAP9

Table 3.1 summarizes the main architectural characteristics of the GAP8 and GAP9 platforms, highlighting their evolution across generations.

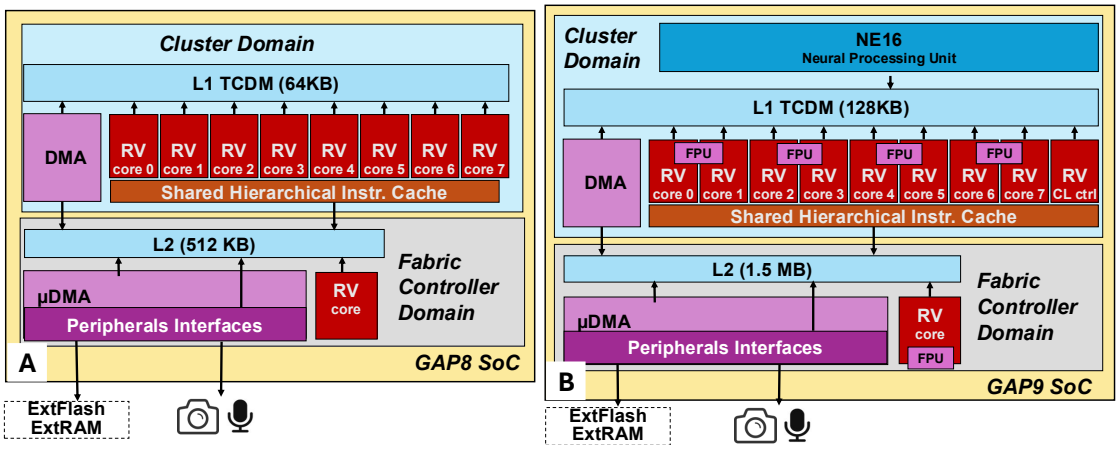

Figure 3.1. Simplified architecture of the GAP8 A) and GAP9 B) SoCs

Table 3.1. Comparison between the GAP8 and the GAP9 architectures.

	GAP8	**GAP9**
RISC-V Cores *Host* + *Cluster*	1+8	1+9
On-chip L1 memory	64 kB	128 kB
On-chip L2 memory	512 kB	1.5 MB
CNN Accelerator	No	NE16
Floating point hardware support	No	Yes
VDD	1.2 V	0.8 V
Max Clock Frequency *Host*	250 MHz	370 MHz
Max Clock Frequency *Cluster*	175 MHz	370 MHz
Fabrication Technology	65 nm	22 nm
Peak Efficiency	4.24 mW/GOP	0.33 mW/GOP

The GAP9 represents a substantial architectural evolution over GAP8. Its increased on-chip memory capacity and bandwidth support larger and deeper neural networks without relying on frequent off-chip memory accesses. The integration of the NE16 accelerator and FPUs extends computational flexibility across both integer and floating-point domains, enabling mixed-precision neural inference at minimal energy cost.

In summary, while GAP8 established the foundational architecture for energy-efficient parallel computing on ultra-low-power devices, GAP9 extends this paradigm toward AI-centric workloads by incorporating dedicated hardware acceleration and

expanded memory resources.

3.4 Deep Learning Compilers for Edge Devices

Deep learning compilers for edge devices are specialized toolchains that translate high-level neural-network representations, typically expressed in frameworks such as TensorFlow, PyTorch, or ONNX [76, 77, 78], into low-level, hardware-specific code, typically C, optimized for execution on resource-constrained microcontrollers and embedded SoCs. For the architectures considered in this thesis, namely GAP8 and GAP9, deployment follows an ahead-of-time code-generation approach coupled with explicit memory management. In this paradigm, the compiler analyzes the network structure before execution and statically determines tiling, buffer placement, and data-transfer scheduling to exploit the hierarchical memory architecture of the target system. This strategy enables fine-grained control over the use of on-chip and, when available, off-chip memory, which is essential for sustaining high throughput within tight memory constraints.

Several frameworks have adopted this design philosophy, differing mainly in how they model and optimize memory allocation and computation scheduling. DORY [79] and Deeploy [80] formulate the tiling and buffer-allocation process as a constraint-programming problem, maximizing on-chip memory utilization under layer-wise resource limits. DORY focuses on static tiling strategies optimized for the small L1 memories typical of microcontrollers, while Deeploy introduces a bottom-up compilation flow that integrates user-defined C kernels and applies code-generation passes to manage tiling and allocation automatically. HTVM [81] and MATCH [82] extend this concept within the TVM ecosystem using the Bring-Your-Own-Codegen (BYOC) interface [83], supporting accelerator-specific code generation for heterogeneous platforms. TelaMalloc [84] tackles the same challenge through a hybrid heuristic-plus-constraint-solver approach that prioritizes layers with the highest buffer concurrency to minimize contention and fragmentation in on-chip memory.

Among these frameworks, AutoTiler [85] generates tiled, memory-optimized C code for the GAP8 and GAP9 RISC-V multi-core SoCs. It orchestrates data movement across the memory hierarchy, leveraging the micro-DMA and cluster-DMA hardware units to reduce overhead for transfers within and between L1, L2, and external memories. As part of the official GAP software stack and specifically tailored to these architectures, AutoTiler is used in this thesis to deploy the evaluated neural networks.

Chapter 4

Object Detection on a Nano-Drone Using a RISC-V Multi-Core MCU

In this chapter, I investigate the feasibility and practical limitations of executing deep neural networks on resource- and energy-constrained parallel microcontroller units (MCUs). Specifically, I focus on the GAP8 platform, a parallel ultra-low-power processor representative of multicore embedded MCUs. The analysis builds upon two experimental studies that collectively explore the trade-offs between computational efficiency, model complexity, and task performance in embedded artificial intelligence.

I begin by examining the deployment of a CNN for object detection, based on the MobileNetV2-SSD architecture. The MobileNetV2 model introduces a width multiplier parameter, commonly denoted as α, which uniformly scales the number of channels in each convolutional layer. This parameter allows balancing computational cost and model accuracy, effectively controlling the network's depth and representational capacity. By varying α across different configurations, $\alpha = 0.5,0.75,1.0$, I evaluate the trade-off between execution latency and detection accuracy when executing on the GAP8 MCU.

The analysis reveals that while lightweight configurations yield faster inference (4 3frames/s for $\alpha = 0.5$ 2.7× faster than $\alpha = 1.0$, their accuracy degradation is non-negligible for perception tasks of practical relevance (13% loss in mAP reaching . Ultimately, the full-width configuration ($\alpha = 1.0$) achieves the most favorable performance-to-accuracy balance under the given hardware constraints, and it is therefore selected as the baseline for subsequent experiments. Building upon this

finding, I integrate the object detection network with an additional model dedicated to obstacle avoidance, demonstrating that the concurrent execution of multiple perception tasks quickly saturates the available computational throughput. This observation underscores the necessity of employing more capable embedded hardware when high-throughput inference or multi-network integration is required, a consideration that becomes particularly critical in autonomous robotic systems where all computation must occur on-board.

The rapid evolution of the Internet of Thingss (IoTs) is driving the emergence of intelligent, ultra-low-power edge devices capable of performing complex tasks in full autonomy [86]. Among these, nano-sized UAVs represent a particularly challenging and promising class of systems. Their small form factor and lightweight design enable safe operation in confined environments and close proximity to humans [35], making them ideal candidates for applications such as indoor exploration, search and rescue [87], and inspection. Achieving full autonomy in such platforms, however, requires integrating high-level perception and decision-making capabilities under stringent energy and computational constraints [88, 89]. Since only a small fraction of the total power budget (5-10%) can be allocated to computation [90], these systems typically rely on low-power microcontrollers, which limits their ability to execute demanding artificial intelligence workloads.

Inspired by the functional mapping of the different regions of human brains [91], I address this problem by mapping multiple tasks on the two MCUs available aboard my nano-drone, GAP8 SoCs, a Parallel Ultra Low Powers (PULPs) multi-core processor and the STM32F405. The STM32F405 executes low-level control, sensor interfacing, and estimation tasks, while the GAP8 acts as a co-processor dedicated to CNNs inference .

The main perception task addressed in this chapter is object detection during autonomous exploration of previously unseen environments. The detection pipeline is based on a Single Shot MultiBox Detectors (SSDs) employing a MobileNetV2s (MbV2s) backbone. The MbV2s architecture introduces a width multiplier parameter, α, which scales the number of channels in each layer to balance computational cost and accuracy. I deploy three detector variants—SSDs-MbV2s-α, with $\alpha = 0.5, 0.75, 1.0$,requiring 193 MMultiply-and-Accumulates (MACs), 358 MMACs, and 534 MMACs, respectively. The networks are pre-trained on the OpenImages dataset and fine-tuned on the Himax dataset collected with the onboard grayscale camera to mitigate domain shift. Quantization-aware training (QAT) is applied to convert the models to 8-bit integer precision, preserving accuracy while ensuring compatibility with the GAP8's hardware support for only integer operations.

The exploration task is carried out in two different modalities. In the first, a *Time of Flights (ToFs)-only* policy uses four single-beam ToFs (front, back, left, right) to produce reliable short-range, line-of-sight range measurements (<4 m) that drive

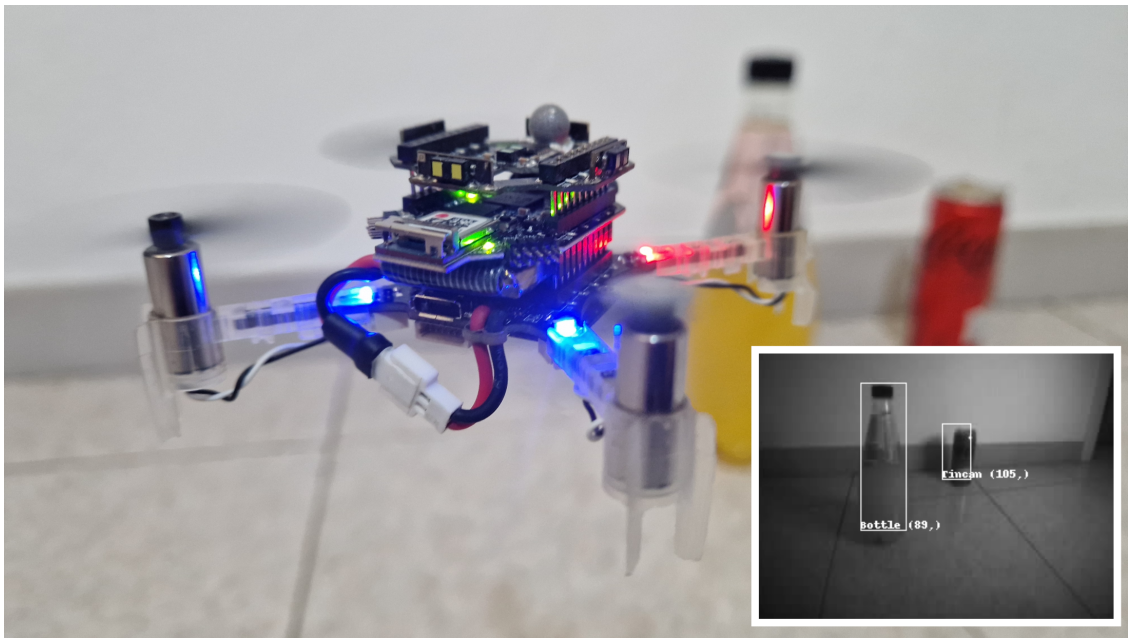

Figure 4.1. My fully autonomous prototype, based on a Crazyflie nano-drone.

a lightweight, reactive exploration controller. In the second, a *mixed* policy fuses ToFs with a front-looking monocular camera processed by *PULP-Dronet* [33], a compact CNN that estimates probability of collision from QVGA grayscale frames; the two cues are complementary—ToFs provide precise near-field ranging while vision extends the perceptual field and adds semantics.

Using my prototype (shown in Fig. 4.1), I first evaluate the baseline system. Under the ToFs-only exploration policy, the drone covers on average 83 % of a 36 m^2 indoor area within 3 min, flying at a mean speed of 0.5 m/s. In the same setting, the vision pipeline based on SSDs-MbV2s runs fully on-board the GAP8: the largest variant ($\alpha = 1.0$) sustains up to 1.6 frame/s at only 134 mW, achieving 50 % mAP on two target classes. Across five independent closed-loop trials, the system recognizes six object instances (two classes) with a mean detection rate of 90 %. To the best of my knowledge, this constitutes the first fully autonomous nano-drone that jointly performs environment exploration, collision avoidance, and real-time object detection relying exclusively on onboard sensing and computation.

I then assess the mixed ToFs+PULP-Dronet modality in a more cluttered environment. The additional visual collision cue, provided by the PULP-Dronet, improves obstacle avoidance robustness (up to +60% over ToFs in dense scenes). However, when this mixed avoidance stack is *interleaved* with the best SSD detector ($\alpha = 1.0$) on the same GAP8, the achievable end-to-end CNNs throughput remains bounded

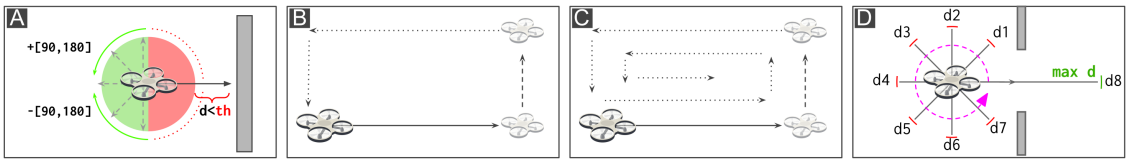

Figure 4.2. Bio-inspired Exploration policies: (A) *Pseudo-random*: inverts the direction by a random angle. (B) *Wall-following*: keeps a fixed distance from the perimeter. (C) *Spiral*: progressively increases the distance from the perimeter. (D) *Rotate-and-measure*: moves along the longest free-space direction.

to 1.6 Hz. In my field tests, this falls short of the $\geq$ 8 Hz rate needed to keep crash-avoidance probability above 80 % at 0.5 m/s. These results indicate that, while multi-sensory fusion markedly improves safety in complex environments, sustaining mixed avoidance *and* high-accuracy detection concurrently calls for novel hardware accelerators that can deliver higher computational throughput within the same power envelope, enabling real-time execution of multiple complex deep learning models.

4.1 Methodology

4.1.1 Robotic platform

The experimental validation of the work presented in this chapter is performed on the Crazyflie 2.1 nano-quadrotor from Bitcraze[1], This compact open-source drone, widely adopted for research in autonomous flight, weighs approximately 27 g and has a diameter of 10 cm. It integrates an STM32F405 MCU for flight control, supported by a Bosch BMI088 inertial measurement unit (IMU) and an nRF51822 MCU for 2.4 GHz communication. Operating at up to 168 MHz with 48 kB of SRAM and 128 kB of flash memory, the STM32F405 manages all low-level control functions, including sensor interfacing, state estimation, flight stabilization, and closed-loop attitude and position control via an extended Kalman filter.

To extend sensing and computing capabilities, the platform is equipped with three additional Bitcraze COTS printed circuit boards (PCBs): the **Flow deck**, the **Multi-ranger deck**, and the **AI-deck** , see Figure 4.3 for a depiction of the main components. The Flow deck provides optical-flow and height measurements through a PMW3901 optical flow sensor and a VL53L1x ToFs module, improving state estimation reliability and reducing drift. The Multi-ranger deck features five single-beam VL53L1x ToFs distance sensors mounted on the drone's top and sides,

[1] `https://www.bitcraze.io/products/crazyflie-2-1`

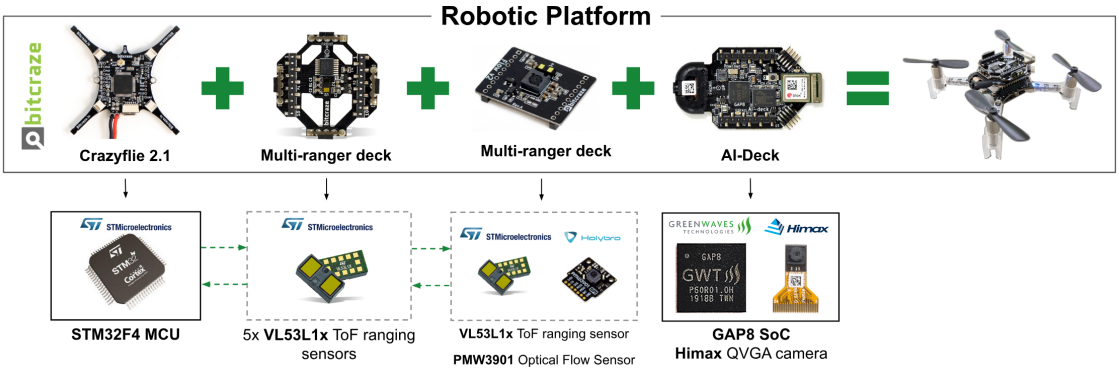

Figure 4.3. The robotic platform structure: Crazyflie 2.1, Flow deck, Multi-ranger deck, and AI-deck.

providing line-of-sight distance measurements in the 0 m to 4 m range at 20 Hz, and is used for obstacle detection and environmental mapping.

Finally, the AI-deck serves as the visual processing unit of the system, extending the drone's onboard computational capabilities. This PCB integrates the **GAP8** system-on-chip (SoC),see section 3.1 for an in depth description, and includes a low-power QVGA grayscale camera (Himax HM01B0) and external memories,8 MB of HyperRAM and 64 MB of HyperFlash, used for buffering and model storage.

4.1.2 Multi-task integration.

I exploit my nano-UAV platform to autonomously explore unknown environments while simultaneously avoiding obstacles and detecting specific target objects. To achieve this goal, I separate the overall mission into three concurrent tasks: an *exploration task*, an *obstacle avoidance task*, and an *object detection task*. Each task is executed onboard, leveraging the computational resources provided by the two MCUs, the STM32F405 and the GAP8.

The *exploration task* is responsible for navigating within an unknown space while preventing collisions and maximizing environmental coverage. The STM32 manages the exploration policy based on a lightweight state machine, as detailed in Section 4.1.3. In particular, the STM32 continuously processes the ToFs measurements provided by the Multi-ranger deck to estimate the proximity of surrounding obstacles. These measurements are used by the exploration policies to determine the next waypoint in terms of forward speed and yaw rate, thus generating the high-level set-points that drive the flight controller.

Collision prevention in my system is implemented in two configurations: a range-based method relying solely on ToFs sensors, and a multi-sensory fusion approach

that combines ToFs and visual information.

In the first configuration, the *ToF-only* mode, the STM32 monitors the distance measurements provided by the Multi-ranger deck and triggers reactive avoidance maneuvers whenever the minimum measured distance falls below 1 m. This configuration enables simple yet effective obstacle avoidance based exclusively on geometric proximity information, without relying on visual inputs.

In the second configuration, the *multi-sensory fusion* mode, I augment the range-based information with visual semantics extracted from a lightweight variant of **PULP-Dronet**, executed on the GAP8 SoC. The network is quantized to 8-bit fixed-point precision, resulting in a memory footprint of 83.9 kB, and achieves a throughput of 63 frames/second on the GAP8. During inference, up to 200 kB of additional memory is used for intermediate activations. The CNN produces a collision probability, which is transmitted to the STM32 via UART. A collision is considered imminent if either the minimum ToF distance is below 1 m or the PULP-Dronet collision probability exceeds 0.7. In this configuration, the STM32 executes the final decision logic, combining both sensory inputs to issue avoidance commands.

Finally, the *object detection task* runs concurrently with the other two, leveraging the GAP8's multi-core cluster to execute the convolutional neural network described in Section 4.1.4. The GAP8 handles both camera acquisition and CNN inference, outputting the pixel coordinates and class labels of detected objects. As a result, the processing is done fully onboard, making the final closed-loop system completely autonomous, i.e., with no external communication or computation.

4.1.3 Exploration algorithms

The four bio-inspired exploration strategies implemented in this chapter are summarized in Fig. 4.2. Each policy uses distance information from three ToF sensors

Table 4.1. Mean Average Precision (mAP) of the SSD CNNs trained on Google OpenImages and finetuned on the Himax dataset.

Testing dataset	**Fine-tuning**	**Format**	**SSD size**		
			1×	0.75×	0.5×
OpenImages	*no*	`float32`	59%	47%	43%
Himax	*no*	`float32`	50%	41%	29%
Himax	*yes*	`float32`	55%	46%	43%
Himax	*yes*	`int8`	50%	48%	32%

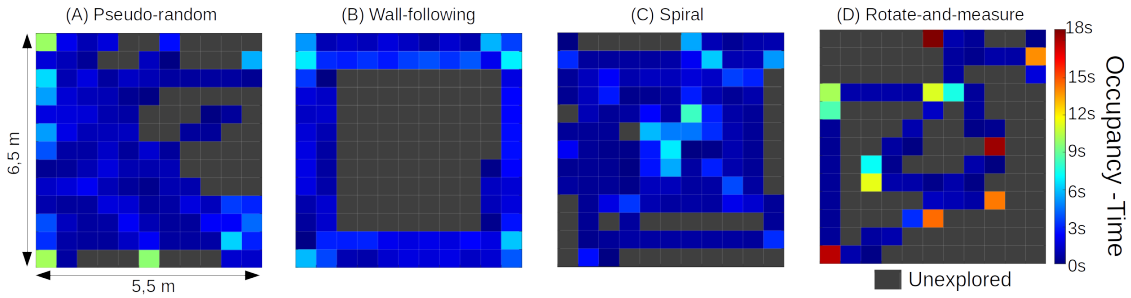

Figure 4.4. Occupancy maps measured using the four exploration policies. During the experiments, the nano-drone flies at 0.5 m/s. The color of any cell, which represents a 0.5×0.5 m area, indicates the time spent by the nano-agent in that area (up to 18 sec). Black is used if the cell has not been explored.

positioned at the drone's front, left, and right sides. To evaluate their effectiveness, I record the drone's motion within a 6.5 m×5.5 m environment equipped with a motion-capture system operating at 50 Hz. The workspace is discretized into 0.5×0.5 m cells, and I visualize the spatial occupancy of the drone over a 3 min flight as a heatmap (Fig. 4.4). The four implemented policies are described below.

A) Pseudo-random. Inspired by the erratic motion patterns often observed in natural organisms [39, 89], this strategy drives the drone forward until an obstacle is detected within 1 m. When this occurs, a random heading change greater than ±90° is applied (Fig. 4.2-A), reducing the chance of revisiting previously sensed obstacles.

B) Wall-following. This algorithm constrains the drone to move along the room boundaries, maintaining a fixed lateral clearance of 0.5 m as measured by the side ToF sensors (Fig. 4.2-B). Upon encountering a front obstacle, the drone performs a ±90° rotation toward a free direction before continuing. By design, this strategy explores only the perimeter, leaving the inner area unvisited.

C) Spiral. Here, the drone performs a sequence of concentric perimetral paths (analogous to wall-following) with a gradually increasing offset from the outer walls (Fig. 4.2-C). Once the center is reached, the pattern is reversed, progressively shrinking back toward the starting position. The initial distance from the walls is 0.5 m, and it is incremented or decremented by the same step after each lap, resulting in an expanding–contracting spiral trajectory.

D) Rotate-and-measure. This policy alternates two phases (Fig. 4.2-D). First, the drone performs an in-place 360° rotation, sampling the front distance every 45°. Then, it proceeds toward the direction corresponding to the maximum measured distance for up to 2 m. This approach promotes exploration of the room's central area, though it tends to leave corners less frequently visited.

4.1.4 Object detection algorithm design

The onboard object detection module adopts an SSD architecture with a MobileNetV2 backbone [25, 92], pre-trained on the OpenImages dataset [93]. The backbone is followed by multiple detection heads responsible for estimating object locations, categories, and confidence scores across feature maps of different resolutions. To balance accuracy against computational cost, I deploy three model variants by adjusting the MobileNetV2 width multiplier α. These configurations are denoted as *SSD-MbV2-*α, where $\alpha = 0.5,0.75,1.0$. The largest model (*SSD-MbV2-1.0*) includes 4.67 M parameters and requires 534 MMAC operations per frame, while the smaller *SSD-MbV2-0.75* and *SSD-MbV2-0.5* variants use 2.68 M and 1.34 M parameters, corresponding to 358 MMAC and 193 MMAC, respectively.

All models are trained to detect two object classes—*bottles* and *tin cans*—using a curated subset of OpenImages. Since the dataset is unbalanced (1306 tin can and 11306 bottle images), additional tin can samples are generated by horizontally translating the originals by up to 10% of the image width. The final split includes 19142 training, 208 validation, and 663 testing samples, consistent with the original dataset partitioning. To mitigate the domain gap between OpenImages and the onboard camera images (Fig. 4.5), I further fine-tune the networks on a custom-collected *Himax Dataset*, composed of 321 training and 279 test images.

Deployment on the GAP8 SoC requires 8-bit quantization. To preserve accuracy, I apply quantization-aware training (QAT) using the TensorFlow Object Detection API [92] with symmetric quantization, matching the integer arithmetic supported by the target hardware. The final models are compiled into C code using GreenWaves' GAP*flow* toolchain[2] limiting the L2 memory allocation to 250 kB.

4.2 Experimental Results

4.2.1 Object detection evaluation

The SSD variants are trained on OpenImages for 1200 epochs using RMSProp. I adopt an initial learning rate of $8 \cdot 10^{-4}$ with an exponential decay of 0.95 every 24 epochs and a batch size of 24. All images are resized to 320×240 to match the onboard camera resolution. Training employs standard photometric augmentations—horizontal flip, brightness jitter, random crop, and grayscale—each applied independently with probability 0.5. Prior to deployment, I fine-tune the SSDs on the Himax dataset for 100 epochs (with QAT enabled), using a learning rate of 10^{-4} and the same 0.95 decay every 10 epochs.

[2]https://greenwaves-technologies.com/gapflow/

Tab. 4.1 summarizes mAP (COCO definition [26]) across model scales on both OpenImages and Himax test sets. On OpenImages, the largest network reaches 59% mAP, up to +16% over the smallest model. When evaluated on Himax without fine-tuning, the scores drop by 9%, 6%, and 14% for the three models, which I attribute to the lower quality of onboard imagery relative to web images. Fine-tuning on Himax recovers performance, with the best model attaining 55% mAP, narrowing the gap to the OpenImages result. Under 8-bit quantization, the top post-QAT mAPs are 50% and 48%, so I retain *SSD-MbV2-1.0* and *SSD-MbV2-0.75* for field tests.

Tab. 4.2 reports runtime metrics on GAP8. I set 1.2 V core voltage, 160 MHz for the compute cluster, and 250 MHz for peripherals. *SSD-MbV2-1.0* sustains 1.6 FPS with 5.3 MAC/cycle. The *0.75*× and *0.5*× variants are 1.6× and 2.7× faster, respectively. Peak AI-deck power is 143.5 mW with *SSD-MbV2-0.75*, where memory bandwidth and compute utilization are best matched; power reduces to 134.5 mW for *SSD-MbV2-1.0*.

Table 4.2. SSD CNNs' onboard performance.

SSD	**Parameters**	**Operations**	**Efficiency**	**Throughput**
1×	4.7M	534 MMAC	5.3 MAC/cycles	1.6 FPS
0.75×	2.7M	358 MMAC	5.9 MAC/cycles	2.3 FPS
0.5×	1.2M	193 MMAC	5.3 MAC/cycles	4.3 FPS

Figure 4.5. Comparison between an OpenImages sample (left) and an image captured by the onboard Himax sensor (right).

4.2.2 Exploration policies evaluation

I quantify each policy's effectiveness by the *coverage area* (%) in the room from 4.1.3, computed as visited cells over the total (143 cells). A cell is marked visited when the drone's center of mass lies within it. Each policy is tested at three mean speeds—0.1 m/s, 0.5 m/s, and 1 m/s—yielding 12 configurations. For each configuration, I run five 3 min flights, for a total of 60 runs (3 h flight time).

Fig. 4.6 shows mean coverage per configuration. *Pseudo-random* and *spiral* benefit most from higher speeds: from 35% at 0.1 m/s to 74%/82% at 0.5 m/s, and to 80%/83% at 1 m/s. *Wall-following* and *rotate-and-measure* modestly improve from 0.1 m/s to 0.5 m/s (+17% and +14%), with minimal change at 1 m/s (+2% and -1%). The perimeter-only nature of *wall-following* limits coverage (see Fig. 4.4-B), while *rotate-and-measure* spends considerable time spinning and favors the central area (Fig. 4.4-D).

4.2.3 In-field object detection and exploration tasks closed-loop evaluation

I assess end-to-end performance by measuring detection rate with three bottles and three tin cans placed in the same, obstacle free see Fig. 4.7, room setup as Sec. 4.2.2 (one of each near the center, fmy near corners). Detection outcomes depend on *i)* detector precision/throughput and *ii)* coverage (affected by flight speed). Tab. 4.3 compares the two best SSDs (Sec. 4.2.1) across all policies and speeds. *SSD-MbV2-1.0* consistently matches or exceeds the *0.75*× model, indicating accuracy dominates throughput in my setting (*0.75*× is faster but has lower mAP).

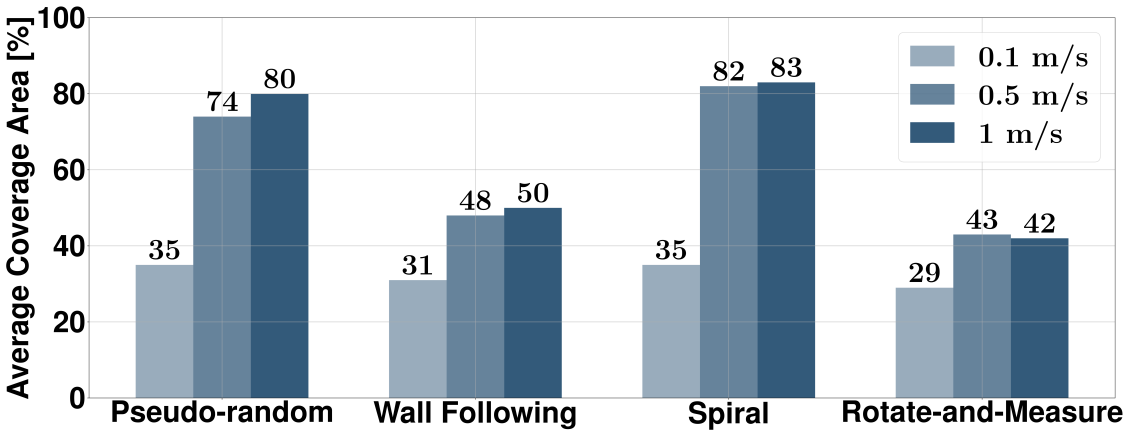

Figure 4.6. Average coverage area (in %) for each exploration policy at mean flight speeds of 0.1 m/s, 0.5 m/s, and 1 m/s.

Table 4.3. Average detection rate: 6 objects, 5 runs of 3 minutes each.

SSD	Flight speed [m/s]	Detection rate			
		Pseudo random	Wall following	Spiral	Rotate and measure
1.0×	0.1	27%	**63%**	67%	**53%**
	0.5	**90%**	50%	**73%**	**53%**
	1	83%	53%	70%	47%
0.75×	0.1	27%	**50%**	33%	47%
	0.5	**80%**	37%	**43%**	**50%**
	1	37%	27%	**43%**	33%

For *SSD-MbV2-1.0*, *pseudo-random* and *spiral* peak at 0.5 m/s. This contrasts with Sec. 4.2.2, where coverage favored 1 m/s; here, 1 m/s likely overwhelms the detector given its 1.6 FPS limit. *Wall-following* and *rotate-and-measure* underperform (max 63% and 53%), consistent with restricted coverage patterns.

Fig. 4.8 details the best configuration: *pseudo-random* at 0.5 m/s with *SSD-MbV2-1.0*. Average coverage over five 3 min flights reaches 72% (variance 21%). A representative run achieves 100% detection in 154 s (blue dots).

Finally, Tab. 4.4 reports the platform power profile, measured with the Power Profiler Kit 2 (Nordic). Running *SSD-MbV2-1.0*, the AI-deck accounts for 1.67% of total power; motors dominate at 91.31%. Crazyflie MCU and ToF sensors consume the remaining 7.03%.

Figure 4.7. Environments for in-field testing. The test arena is 6.6x5.6m^2 wide, restricted to 1.95x4.5m^2 for the narrow-corridor scenario.

Table 4.4. Power breakdown of the robotic platform.

	Motors	CF elect.	AI-deck	Multi-ranger	**Total**
Power [W]	7.32	0.277	0.134	0.286	**8.02**
Percentage	91.31%	3.45%	1.67%	3.57%	**100%**

4.2.4 Multi-sensory collision avoidance

I compare ToF-only collision avoidance with a combined ToF&PULP-Dronet approach. Fig. 4.7 depicts three environments: (A) obstacle-free, (B) obstacle-populated, and (C) narrow corridor. Environments B/C include thin obstacles (chair legs, tripods); C further introduces walls forming a 2×4.5 m corridor. Flights start from a known pose and use the pseudo-random policy (Sec. 4.1.3). I collect 5 and 10 runs in A and C, and 20 runs in B, at 0.5 m/s and 0.3 m altitude. Trajectories are motion-capture tracked; collisions are manually annotated.

Table 4.5 lists success rates (collision-free missions), average collisions per minute, and average coverage per minute. I also include a single-agent baseline from the SniffyBug swarm [94], which relies on ToF. In the obstacle-free case, ToF suffices (100% success), so PULP-Dronet was not tested. With clutter and narrow passages, ToF-only and SniffyBug struggle due to missed detections of thin obstacles (20%/15% success in B; 0% in C). The ToF&PULP-Dronet system reaches 80%

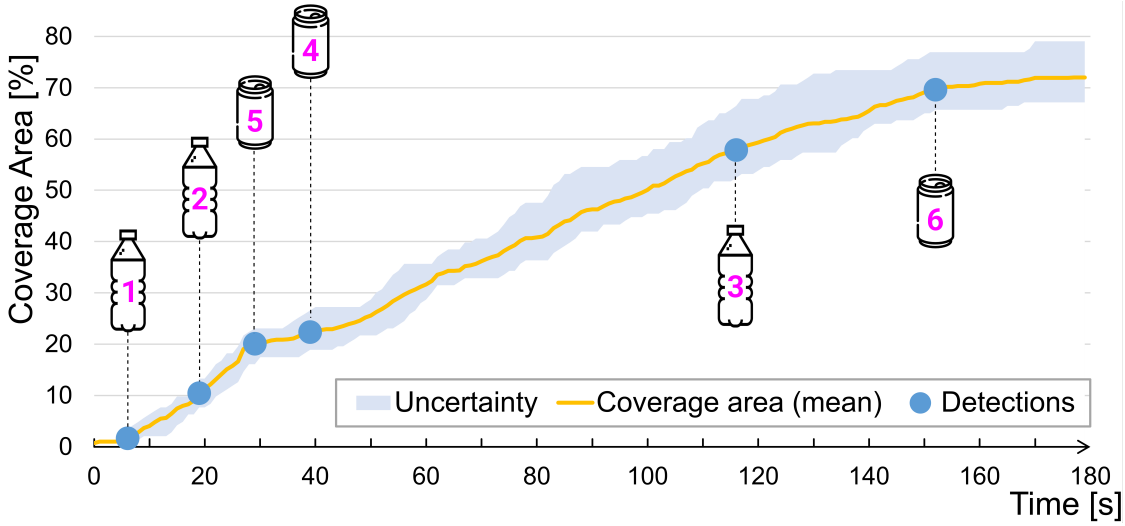

Figure 4.8. Coverage area of the *random* policy over 5 runs (mean and variance). Detection times for 6 targets (blue dots) are from a single run.

Table 4.5. In-field obstacle avoidance performance exploration.

Env.	Parameters	SniffyBug	ToF-only	ToF&PULP-Dronet
obstacle free	Crash-free rounds	5 /5	5 /5	-
	Avg. crash/min	0	0	-
	Avg. coverage/min[%]	20.7	22.4	-
populated	Crash-free rounds	4 /20	3 /20	16 /20
	Avg. crash/min	1.8	1.5	0.2
	Avg. coverage/min[%]	33.2	22.6	10.3
narrow corridor	Crash-free rounds	0 /10	1 /10	7 /10
	Avg. crash/min	2.6	2.4	0.4
	Avg. coverage/min[%]	19.8	45.2	26.7

(B) and 70% (C) success. In the narrow corridor, ToF-only attains higher coverage/min (45.2%) than the combined system, as the latter slows to avoid predicted collisions. Overall, augmenting ToF with CNN-based vision improves success in cluttered settings from 20% to 80%.

4.2.5 Multi-tasking AI execution

Integrating PULP-Dronet (Sec. 4.1.2) with the best detector (*SSD-MbV2-1.0*, Sec. 4.2.3) yields an interleaved throughput of 1.6 frame/s on GAP8. Table 4.6 details throughput, parameter size, L2 use, and power for the exploration logic (STM32) and both CNNs (GAP8). In this configuration, PULP-Dronet's effective contribution is throttled by the detector's rate, which can degrade the gains observed in Sec. 4.2.4.

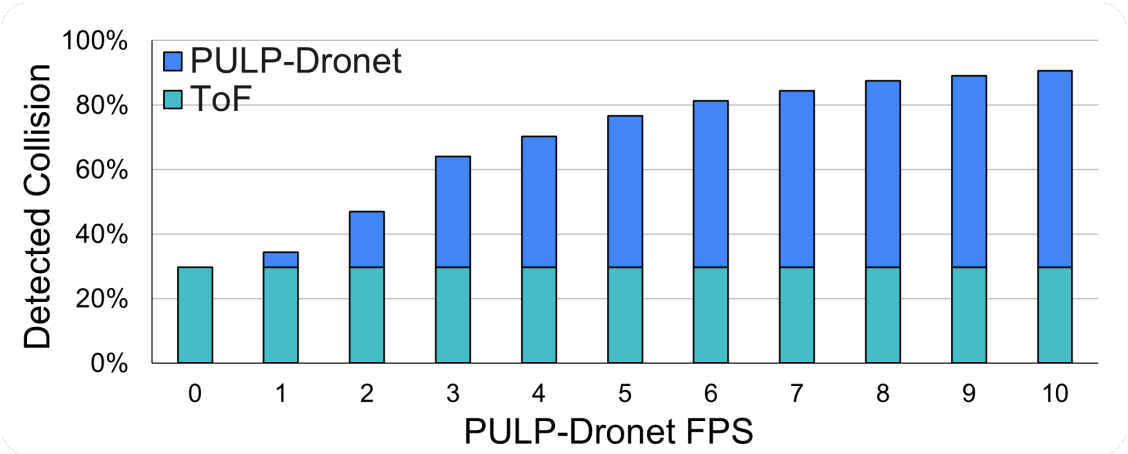

Figure 4.9. PULP-Dronet obstacle detection at varying throughput.

Table 4.6. Embedding cost for the multi-task integration. The Parameters are stored in the Flash Memory. Memory refers to the maximum usage of the on-chip L2 memory.

MCU	Task	Throughput [Hz]	Parameters [MB]	Memory [KB]	Power [mW]
STM32	Exploration policy	20	-	0.1	286
GAP8	SSD	1.7	4.67	250	134
	PULP-Dronet	63	0.08	200	101
	SSD&PULP-Dronet	1.6	4.75	250	133

To analyze throughput–avoidance trade-offs, I recorded 64 videos (10 FPS) of approaches to 8 obstacles (6 chairs, 2 tripods) from 60 cm at 0.5 m/s, with at least 12 frames per clip and synchronized ToF. I sub-sample to emulate 1 Hz–10 Hz PULP-Dronet rates and evaluate detection outcomes (Fig. 4.9). The ToF-only baseline (0 FPS) is constant across settings. At 2 FPS, PULP-Dronet yields +17% over ToF-only; between 3–6 FPS, gains rise from 34% to 51%; benefits taper at higher rates, with 87.5% at 8 FPS and a peak of 90.5% at 10 FPS.

In summary, I demonstrated concurrent deployment of two CNNs on a nano-UAV under tight MCU-class memory budgets. While this enables multi-task intelligence reminiscent of tiny biological systems, the joint throughput remains bounded to 1.6 FPS. Further gains would require either a more capable MCU or additional optimization of the dominant bottleneck, object detection, currently requiring 573 ms per inference.

4.3 Conclusions

This chapter has demonstrated that current multi-core MCUs are capable of executing complex workloads, such as object detection and concurrent neural inference, within strict memory and power constraints. The nano-UAV experiments show that such computational capabilities are sufficient to sustain multiple perception and control tasks directly onboard, thereby enabling complete autonomous operation without external computation.

Nevertheless, the results also highlight that timing remains a critical limitation: tasks strongly dependent on inference throughput, such as real-time visual perception, are bounded by the available computational capability. Addressing this constraint will require the adoption of more powerful and specialized hardware, together with a tighter hardware–software co-design approach to sustain higher-speed execution while preserving energy efficiency.

Chapter 5

Heterogeneous MCU for On-Device Pest Detection

In the previous chapter, I analyzed the capabilities of modern multi-core MCUs, showing that these devices can sustain increasingly complex workloads, including convolutional neural networks, within tight memory and power budgets. Nevertheless, the evaluation also revealed the limitations that arise when executing compute-intensive tasks entirely on general-purpose cores, particularly in applications where latency is critical.

To overcome these constraints, recent embedded platforms have adopted a heterogeneous architecture that integrates domain-specific hardware accelerators alongside conventional processing cores. Such accelerators range from classical functional units, such as Floating Point Units (FPUs), to specialized computing engines optimized for machine learning workloads. These heterogeneous systems offer substantial gains in performance and energy efficiency, enabling the execution of complex algorithms that were previously infeasible on baseline MCUs.

In this chapter, I investigate this new class of processing systems through the case study of an automated pest detection sensor for precision agriculture. I target the detection of the codling moth (*Cydia pomonella*), a major threat to orchard crops, and employ the GAP9 SoCs as a representative heterogeneous MCU featuring both general-purpose RISC-V cores and a dedicated neural acceleration engine.

The codling moth pest, scientifically known as *Cydia pomonella*, poses a significant threat to global crop production, potentially causing up to 80% fruit abscission in apple orchards [95, 96]. To prevent an excessive utilization of pesticides, it is essential to promptly detect the threat and perform localized treatments only in the affected areas [97, 98]. Given the size of the areas occupied by the orchard fields,

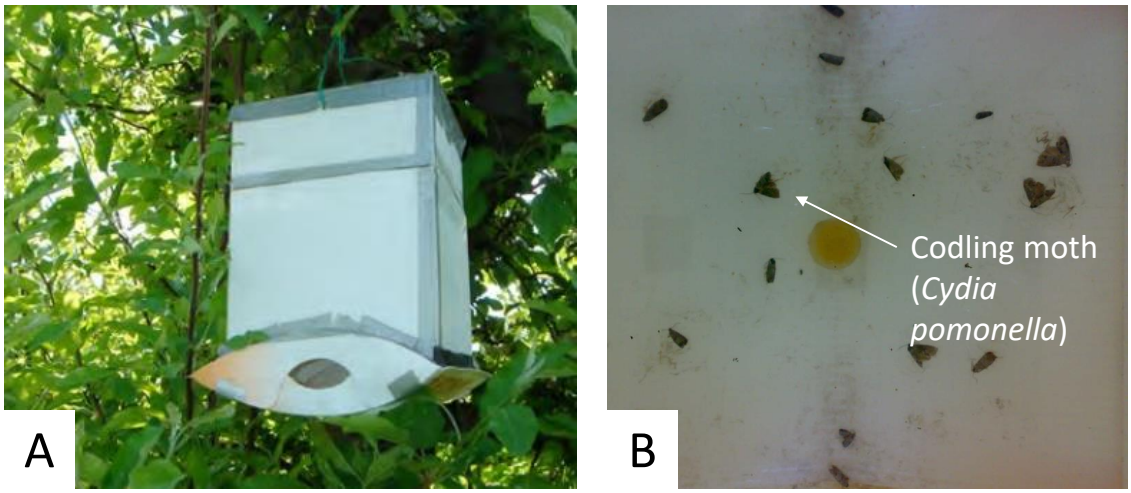

Figure 5.1. A) Trap prototype for codling moths proposed in [1] and B) an image sample taken from a trap deployed in the field [1, 2].

novel automated pest detection systems have become crucial to replace costly and time-consuming human inspection [99].

Fig. 5.1-A shows a trap commonly used for codling moth detection [1]. The prototype contains a sticky pad with a sex pheromone bait to attract the pests. An internal camera records the trapped insects and sends the images (e.g., Fig. 5.1-B) to a remote server for analysis. The presence of the pests is assessed by detecting the codling moths using computer vision algorithms [100].

Because of the nature of the installation environment, the sensor system can only be powered by batteries or energy harvesters, e.g., solar panels. This requirement severely constrains the energy the individual electronic components like the image sensor, the central processing unit, and the communication module can consume. In particular, long-range transmissions of images typically require power-hungry interfaces, such as a Global System for Mobile Communications (GSM) subsystem. As reported by *López et al.*, the GSM interface consumes 1.5 Watts in the active state, which accounts for 66% of the total system power [101], thus significantly impacting the lifetime of the battery-operated sensor (if I consider one image per hour it would account for ∼20 months of operation of the sensor).

Thanks to this *near-sensor processing* approach, only a few bytes of information reporting the pest counts are occasionally sent to the remote station. The low data rate enables the usage of ultra-low power protocols such as LoRa [102], thus reducing the communication power cost compared to naive *sense-and-transmit* systems [44].

On the processing side, Microcontroller units (MCUs) are commonly selected as

the core devices of low-power sensor nodes [103]. These devices include single or multiple CPU cores clocked at up to a few hundred MHz and up to a few MB of on-chip memory. In my recent work [50], I showed an optimized Viola-Jones detector for pest detection on the GAP8 System-on-Chip (SoC), an energy-efficient MCU featuring a compute cluster with eight general-purpose RISC-V cores. This method achieves a processing latency 51% lower than a previous computer vision pipeline running on the same SoC [2], which combined a background subtraction algorithm to localize the pests with a Convolutional Neural Network (CNN) for classifying the detected objects. This thesis extends my precedent study [50] by analyzing traditional vs. CNN-based image object detection algorithms for on-device pest detection on a State-of-the-Art (SoA) MCU featuring a convolution accelerator. This class of processing devices, not yet considered in the context of pest-detection systems, has been recently introduced to speed up onboard analytics tasks by 10-100× with respect to utilizing only a single general-purpose core (e.g., ∼56× in [104] for CNN inference). **I selected the GAP9 SoC, an energy-efficient MCU that combines 10 general-purpose cores with a dedicated hardware engine for CNN processing, as an example of this novel class of heterogeneous platforms**. I compare the efficient execution of my optimized Viola-Jones detector with a CNN-based image object detector, namely MobileNetV3-SSDLite (MBNV3-SSD), featuring 3.44 million parameters and a computational load of 584 M Multiply-Accumulate (MAC) operations. Finally, the processing unit is coupled with a low-power image sensor and a LoRa transmission module.

In summary, this thesis makes the following contributions:

- I compare a traditional Viola-Jones detector vs. a CNN-based approach for detecting *Cydia pomonella*;
- I analyze latency-optimized versions of the two methods when running on the SoA GAP9 SoC;
- I evaluate the system-level costs of the pest detection sensor using a duty-cycling power management scheme.

My study shows that the Viola-Jones detector and MBNV3-SSD reach a similar detection rate of, respectively, 81.7 % and 83 % on a first pest dataset. For a second set of images with smaller-sized insects, the CNN-based detector can generalize better than Viola-Jones, achieving a detection rate of 72 %, +27 % than the other method. On the GAP9 SoC, my Viola-Jones algorithm completes the detection task in 221 ms over a 320×240 pixel image, which is 1.47× faster than the execution time on GAP8, thanks to the higher clock speed (+37 %) and the larger memory (+64 kB of low-level memory) of the most recent processing unit. On the other side, the MBNV3-SSD execution takes 147 ms when using the convolution accelerator, 9.5×

faster than on the GAP8, which only uses the available general-purpose cores to run CNN workloads. This speed-up is mainly explained by the higher inference efficiency of GAP9 vs. GAP8, achieving a performance level of 10.9 MAC/cycle (7.2× higher than GAP8). Thus, my study indicates that the GAP9 heterogeneous platform can enable on-board processing of accurate CNN-based pest detectors, which was previously shown only on processing units with more than 15× higher power consumption [20]. From a system-level perspective, I estimated a battery lifetime of ∼199 days if the system is powered by a 1000 mA h battery at 3.7 V and an image is analyzed every 30 seconds with a duty-cycled power management scheme.

5.1 Background

5.1.1 Viola-Jones

The Viola-Jones algorithm [105] scans an image with a moving window and checks for the presence of the target objects. The algorithm extracts the hand-crafted Haar-like features for every image window by applying a set of rectangular filters and then runs a multi-stage cascade classifier. The classifier consists of many simple and computationally inexpensive tests, denoted as *weak classifiers*, which operate on the Haar-like features. Every stage of the cascade classifier includes multiple weak classifiers contributing to a final stage's score. If this score is below a threshold, the detection response on the current window is negative, and no detection is performed. On the other side, a detection is made if all the cascade stages are successfully passed. The set of filters of every stage and the threshold values are determined by a training process fed by positive and negative samples. Instead of operating on the image space, the original paper proposed an intermediate representation, denoted as the *Integral Image* II, to compute the Haar-like features efficiently. This integral image stores at every location (x, y) the sum of the pixels of the input image, i.e., the total brightness level, in the rectangular area delimited by (0,0) and (x, y). For example, in figure 5.2, the pixel A in the integral Image, $II[A]$, is the sum of the pixels in the area A of the input image. By precomputing the integral image, I can easily calculate the integral of any arbitrary area in the input image, which is the founding operation for the Haar-like features.

For example, the integral of the brown area in figure 5.2, is simply computed as $II[D] + II[A] - II[B] - II[C] = 156 + 16 - 40 - 68 = 32$: the operation requires only 4 read accesses in the memory independently of the size of the integral area.

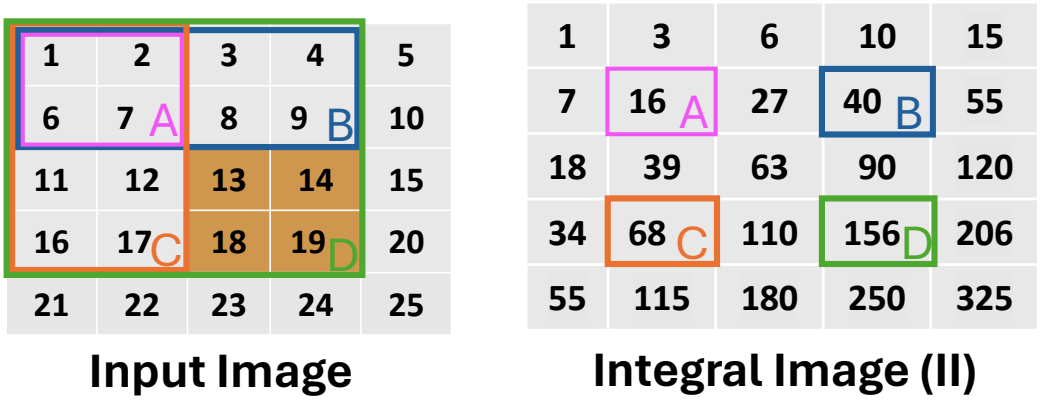

Figure 5.2. Example of the input image (left) and the corresponding integral image (on the right). The values A-D of the integral image correspond to the integral of the areas A-D in the input image (rectangles with matching letters and colors). By precomputing the integral image, the integral value of the brown area can be computed as II[A] -II[B]-II[C] +II[D].

5.1.2 MobileNetV3-SSDLite

Following the analysis performed by [106], I selected the MobilenetV3-SSDLite model for the pest detection task as it represents the best compromise between accuracy and network complexity. Other models with higher accuracy scores cannot fit the memory resources (e.g., Retina-Net [107]) of my platforms or do not feature predictable execution times (e.g., Faster R-CNN [21]), which is essential for real-time embedded devices. MobileNetV3-SSDLite is a CNN-based object detector that is composed of a feature extractor, also denoted as the *backbone*, and a regression (or classification) head [22]. More in detail, this model uses MobileNetV3 [108] as the feature extractor and the SSDLite as the regression head [22]. Both the subnetworks consist of multiple convolution layers. Each layer applies a convolution filter over an input feature map, typically denoted as the *activation* tensor. The activation tensor output is then passed to the next layer. Hence, the final network result is obtained by propagating data through all the layers, also called the *inference* task.

The MobileNetV3 and the SSDLite blocks rely on depthwise-separable convolution as the fundamental operator. Concerning standard 2D convolutional filters, a depthwise-separable convolution is composed of a depthwise and pointwise convolution, reducing the number of filter parameters and operations by 86.8% in the case of a 3×3 convolution with 16 input and output channels. More in detail, the MobileNetV3 architecture includes Inverse Residual Blocks, which expands the number of input features with a pointwise convolution. The produced feature maps

are then passed to a depthwise convolution and finally compressed back with another pointwise layer. A residual connection is added when the input feature size matches the size of the output tensor. On the other side, SSDLite comprises four depthwise-separable convolution blocks, each attending a diverse object size. This regression head takes the features produced by the backbone and produces the bounding boxes and classes of the object inside the image.

5.2 Methodology

5.2.1 On-device Processing Requirements

To determine the throughput requirement for on-device pest detection, I examine the operational role of a smart trap deployed in the field. Smart traps are instrumental in relaying data on pest activity, which is crucial for implementing timely and effective pest control measures. Based on this, I define two application scenarios dictating the latency requirement for the processing task:

- **Low-energy scenario.** In this scenario, I perform detections at the same frequency as pheromone sprays, one every 15 minutes [109], ensuring my platform's responsiveness while minimizing energy consumption. Thanks to this, I can provide information on pest activity, which can be used to optimize pheromone spraying, releasing higher concentrations only when pest activity is detected and reducing unnecessary pheromone waste, as proposed in [110].

- **High-frequency scenario.** In this scenario, I increase the detection frequency to one every 30 seconds to address the limitations of current pest management methods, which rely on daily assessments of pest presence and temperature [111, 112]. This higher detection rate fills a data gap, allowing for more accurate analysis by correlating pest activity with environmental factors like airspeed and humidity. Since the presence of even a single moth can trigger pest disruption methods [113], the capability to detect each new moth entering the trap is necessary to improve my understanding of pest behavior and improve current pest management techniques.

In both scenarios, the system must immediately transmit the pest count after detection to activate the pheromone dispensers. From a system-level viewpoint, I target a battery-powered node that can operate for the whole activity period of the Cydia pomonella, around 4 months [112]. Due to its cost-effectiveness and low cost (less than 10 dollars), I have selected a 1000 mAh battery as a system requirement because this type of battery is commonly used in low-power sensor nodes [114, 115] and IoT sensor nodes [116].

5.2.2 Accelerating Pest Detection on GAP9

This section describes the optimized deployment strategies of the pest detection algorithms on GAP9; for the complete description of the platform, please refer to Sec . 3.2. Both the Viola-Jones and the MBNV3-SSD CNN detectors are trained on a custom dataset composed of 158 images (320×240) with a total of 1275 target insects manually labeled. All the images were taken using an RGB sensor with a resolution of 2048×1536 pixels under daylight conditions (from 8 am to 7 pm) without employing any external illumination. Because the collected images contain 27 non-target objects (<1.6 % of the total amount of pests), of which only 3 in the test split, my detection algorithms are designed to identify only objects belonging to the *Cydia pomonella* category.

Viola-Jones

A 15-stage cascade classifier is trained using a variant of the AdaBoost algorithm [105]. The training set is composed of the 20×20 patches of *Cydia pomonella* cropped from the available images and a set of negative crops from a different image dataset available online[1]. To account for the diverse sizes of the target objects, I run the Viola-Jones algorithm over five scales of the original images. At every iteration, the image resolution is scaled down by a factor of 1.1 in horizontal and vertical dimensions. Detections larger than 30×30 pixels are discarded.

To run the Viola-Jones detector on the GAP9 MCU efficiently, I use the parallel software code presented in my recent work [50]. In this implementation, the input image is stored in the L2 memory, occupying 76 kB (320×240 `INT8`). Unfortunately, the L1 memory is not large enough to fit the input data and the integral image, which demands a total of 307 kB (320×240×4 byte/elements – `INT32` values). To solve this problem, I split the original image (and the scaled versions) into tiles of size 100×240 pixels. Every tile is copied from the L2 to the L1 memory at runtime and then processed. To account for potential detections between adjacent tiles, I set an overlap of 20 pixels in the tiling logic for both the horizontal and vertical directions.

The processing kernel takes an input data tile to compute a tiled version of the integral image, which features a size of 96 kB and is stored in the L1 memory. The CL cores then scan (in parallel) different regions of the image tile with a 20×20 sliding window. Every core invokes the cascade classifier to test if the current window includes the target object. When a stage of the classifier returns a negative response, the evaluation continues to the next window of the tile. The

[1] `https://pythonprogramming.net/static/images/opencv/negative-background-images.zip`

parallelization is operated over eight cluster cores; the ninth core works as the task dispatcher. At the end of the analysis, the algorithm outputs a list of detections described by their bounding box coordinates.

Convolutional Neural Network

The MobileNetV3-SSDLite object detection network features 3.44 M parameters and accounts for 584 M MAC operations to process a 320×240 image. Similarly to Viola-Jones, this detector outputs a set of bounding boxes, each associated with a predicted class, i.e., *Cydia Pomonella* vs. *background*. The model is trained for five epochs using my custom dataset with an 80-20% split between training and validation, resulting in 127 and 31 images, respectively. Stochastic Gradient Descent (SGD) is the optimizer with a momentum of 0.9, a weight decay of $5 * 10^{-4}$, and a learning rate of 0.005. The weights of the MobileNetV3 backbone are initialized from a model pre-trained on the COCO dataset [26]; the parameters of the SSDLite heads are instead learned from scratch. To deploy the CNN detector on GAP9, I use the *GAP*flow toolset[2]. This software package takes the CNN model description (`ONNX` format) and generates the inference `C` code. To use the NE16 accelerator, the model is first quantized to 8-bit using the post-training quantization routine of the *GAP*flow, which takes a few samples from the training set (2 in my case) to estimate the quantization range, i.e., the *calibration* process. The 8-bit quantized parameters are then stored in the external FLASH memory.

At runtime, the inference program follows a layer-by-layer processing schedule. For this task, I allocate one memory buffer in the L2 memory to store the layer operands and a buffer in the L1 memory, serving to store temporary results. The sizes of these buffers are, respectively, 1.2 MB and 115.6 kB. Depending on the layer requirement, the program can transfer the weight parameters from the external FLASH memory to the on-chip L2 memory before the layer execution. Conversely, a layer's input and output activation tensors may be allocated in the on-chip L2 fast memory or the external RAM.

Every layer function copies data (weights and activations) from the L2 (or external) memory to the L1 buffer. It dispatches the processing job, e.g., a convolution operation, on the NE16 engine or the general-purpose cores. The latter option is selected if no convolution accelerator is available, like in the GAP8 SoC, or if the accelerator does not support that layer. I also highlight the impact of the L1 and L2 buffer sizes on the inference throughput. Generally, a larger L2 memory buffer favors allocating more tensors on the on-chip memory, leading to a faster read and write time than accessing data to/from off-chip memories. At the same time, an

[2]https://greenwaves-technologies.com/tools-and-software/

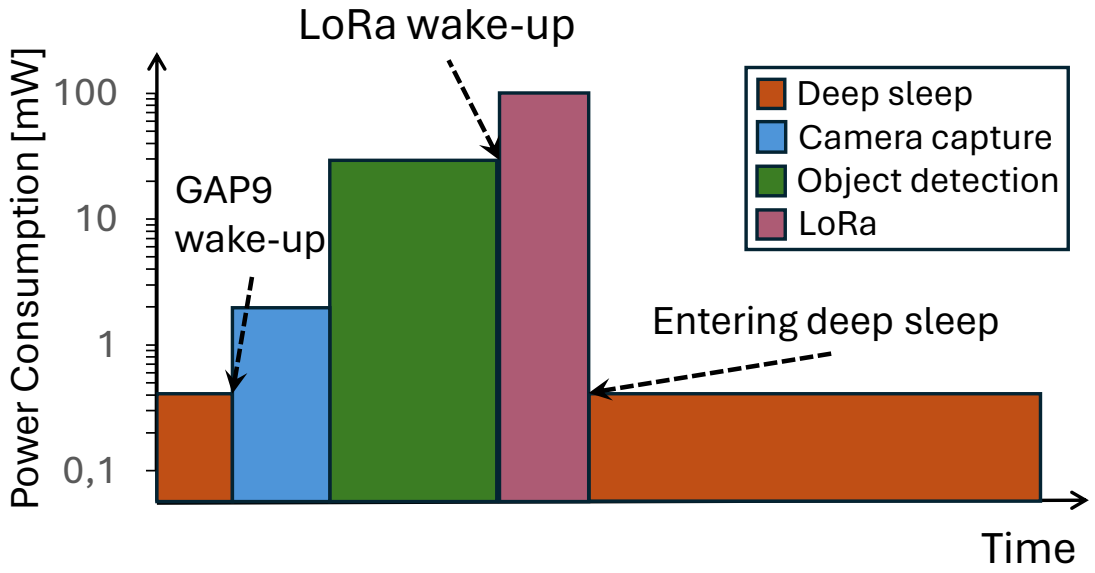

Figure 5.3. System-level power management strategy. The system periodically wakes up from deep sleep mode to capture and process an image. Then, the output is sent via LoRa before going back to deep sleep.

increased L1 buffer reduces the number of copies from the L2 memory, lowering the associated latency overheads.

5.2.3 System-level Power Management

The final pest detection system includes the GAP9 processing unit, a Himax HM01B0 ultra-low-power camera sensor, which captures 320×240 pixels 8-bit grayscale images, and a LoRa module, i.e., the HTCC-AB01 board by CubeCell, connected via UART with the MCU. The latter relies on the AM01 module, which incorporates a low-power cortex M0 MCU with 16 kB of RAM running at 48 MHz and the SX1262 transceiver implementing the physical frequency modulation of the LoRa standard.

Fig. 5.3 shows the used power management scheme to minimize the system energy consumption. Following a duty-cycling scheme, the MCU wakes from deep sleep using the internal real-time clock (RTC). In the active state, the system captures an image and runs the pest detection algorithm. Lastly, the GAP9 MCU activates the HTCC-AB01 board module to transmit the count of detected insects using the LoRa protocol at an energy cost of 1 mJ per byte sent. Once the transmission is completed, the GAP9 receives an acknowledgment signal from the HTCC-AM01 module and returns to the deep sleep mode. I use this same execution scheme for both my low-energy and high-frequency scenarios.

Table 5.1. Detection accuracy on the *near-ds* and *far-ds* datasets.

Method	Quantization	*near-ds*	*far-ds*
Viola-Jones	`int8`	81.7 %	45.0 %
MBNV3-SSD	`float`	88.0 %	84.0 %
MBNV3-SSD	`int8`	83.0 %	72.0 %

5.3 Experimental Results

5.3.1 Detection Accuracy

I tested the effectiveness of the Viola-Jones and the MBNV3-SSD detectors on two test sets, namely *near-ds*, which includes 11 samples with 196 target objects, and *far-ds*, composed of 7 images for a total of 158 objects. The two datasets differ by the distance between the sticky pad and the camera sensor. All the images were taken using an RGB sensor with a resolution of 2048×1536 pixels. For a fair evaluation of the characteristics of the low-power sensor of my system, I resize the images of both datasets to 320×240 pixels and apply a greyscale transformation with additive Gaussian noise to mimic the image quality of the Himax camera sensor.

Tab. 5.1 reports the detection accuracies scored by Viola-Jones and MBNV3-SSD when in full-precision (`float`) or quantized to 8-bit (`int8`). My metric assesses a correct detection if the predicted bounding box overlaps a manual annotation (the IoU score used by the COCO metrics [26] is set to 0.01). The Viola-Jones algorithm achieves a detection accuracy of 81.7% on the *near-ds* but only scores 45% on the *far-ds* dataset. I motivate this gap with the poor generalization capacity of the Viola-Jones algorithm: the training set shows a higher correlation with the *near-ds* than the *far-ds* set. On the other side, the `float` baseline of the MBNV3-SSD detector reaches a superior detection accuracy compared to Viola-Jones in both datasets: 88.0% and 84.0% for, respectively, the *near-ds* and *far-ds*. The post-training quantization process introduces an accuracy drop, reaching −12.0 % for the *far-ds* case. On the other dataset, the degradation in performance is limited to −5 %, possibly also explained by the reduced difference between the training and test domains. As an illustrative example, Fig. 5.4 shows the outputs of the `float` vs. `int8` models on reference samples from the two sets, highlighting the miss-detections of the quantized model.

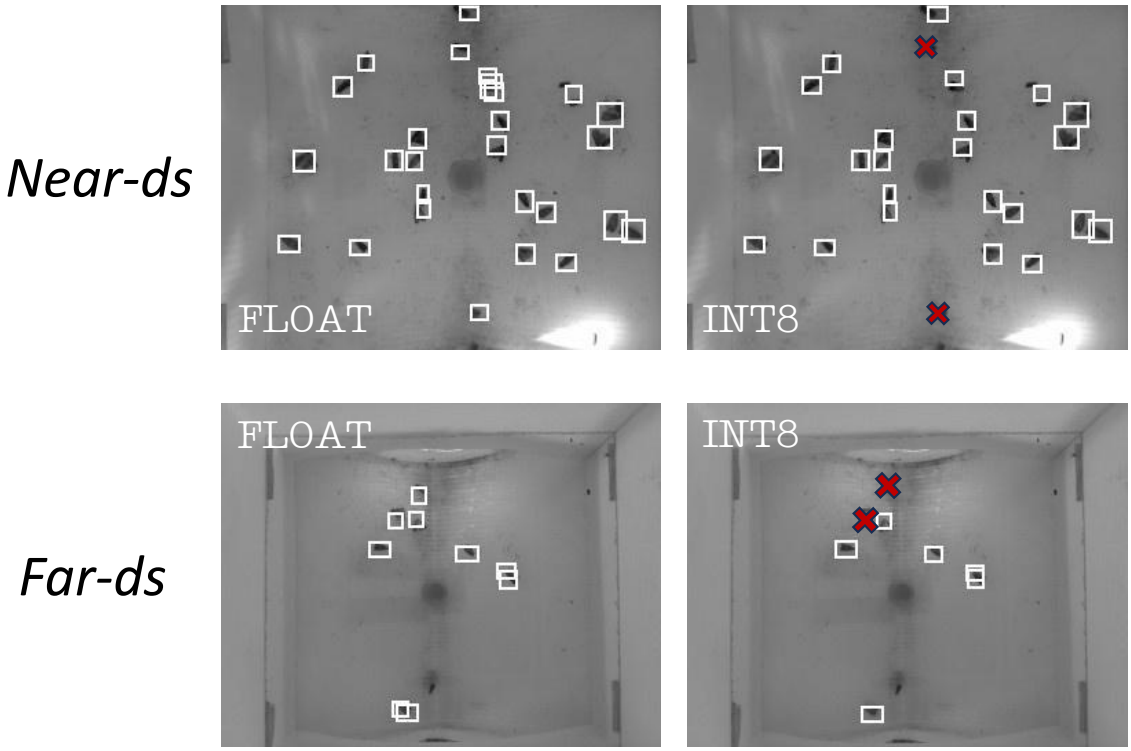

Figure 5.4. CNN-based pest detection on the image samples from the *near-ds* (upper row) and *far-ds* (bottom row) testsets. Outputs from the baseline `float` model (left) are visually compared with the detections of the `int8` quantized model. The red crosses mark the miss-detections.

5.3.2 Latency Analysis

Tab. 5.2 reports the memory usage and the latency measured when executing the Viola-Jones and MBNV3-SSD on the GAP9 SoC with the NE16 accelerator. For comparison purposes, I also report the measurements on the GAP8 MCU, where the CNN inference task runs on the eight general-purpose cores of the cluster (indicated by *CL* in the table). Both chips are configured in their best energy-efficient point: the clock frequency of GAP9 is set to 240 MHz with an operating voltage of 0.65 V while GAP8 operates at 175 MHz at 1.2 V.

The Viola-Jones algorithm requires 384 kB of L2 memory for the input and the integral image. The L1 memory stores an image tile (the tiles size is 100×240 on GAP9 vs. 100×120 on GAP8), with a maximum occupation of 99.6 kB and 51.6 kB, for respectively GAP8 and GAP9. The parameters of the cascade classifier, also preserved in the low-level memory, occupy only 3.56 kB. No external memory is required. On the other hand, MBNV3-SSD uses the off-chip FLASH memory to permanently store the network parameters for a total of 3.44 MB. As described

in Sec. 5.1.2, I allocate an L2 buffer for (part of) the parameters, the activation tensors, and an L1 array acting as the working memory. Additionally, an array of 1.6 MB is allocated in the external RAM for the input and output activation tensors not fitting into the L2 memory.

Tab. 5.2 also reports the number of clock cycles spent for the algorithm executions. For the CNN, I also detail the ratio between MAC and clock cycle (MAC/cycle in the table) as an efficiency indicator. Overall, the execution is 1.43× and 9.5× faster on GAP9 than executing, respectively, Viola-Jones and MBNV3-SSD on GAP8. On the latter device, my parallel Viola-Jones software outperforms the CNN execution by 4.43×. Thanks to the CNN accelerator, GAP9 achieves a processing time of 147 ms for the MBNV3-SSD execution, 1.5× lower than the Viola-Jones latency. Fig. 5.5 gives insights into the CNN inference latency measurements on GAP8 and GAP9 when varying the sizes of the L2 and L1 buffers and when using the general-purpose cores or the NE16 accelerator as the principal compute engine (indicated, respectively, with CL and NE16 in the figure). In the plot, I break down the processing time with respect to the memory locations of the operands of the different layers. More in detail, I distinguish between the total latency of the layers whose inputs and outputs are in the L2 memory (indicated as L2 on the plot, in blue) and the layers that fetch input data from the external memory, highlighting the cases that use 1D or 2D memory transfers, respectively in orange and grey.

When comparing the GAP8 vs. GAP9 execution in the CL setting under the same memory budget (46.7 kB of L1 and 267 kB of L2), the higher clock cycle count on GAP8 is due to not-optimized 2D memory transfers from the external memory (4.6× slower read time). When increasing the memory sizes of the L2 and L1 buffers on GAP9, a 1.4× speed-up is achieved mainly thanks to the higher bandwidth of the on-chip vs. the external memory. In this case, the workload of the layers whose inputs are in the L2 memory dominates the total number of operations (93% of the total) and the total execution time (79% of the clock cycles). These layers' MAC/cycle ratio also increases by +10% on average concerning the low-memory configuration due to the larger tensors stored in the L1 memory buffer. Lastly, the NE16 accelerator further boosts the system efficiency by 1.7×. The speed-up is lower than the theoretical peak because of two main reasons. The non-linear *Hsigmoid* function featured by MobileNetV3 is not natively supported by the accelerator. Hence, it must be executed on the general-purpose cores. Second, the depthwise convolutions underuse by 1/16 of the NE16 compute engine.

5.3.3 Energy Analysis

In Fig. 5.6, I estimate the energy consumption of my system during an active cycle that includes the camera capture, the object detection, and the LoRa wireless

Table 5.2. Memory Footprint and latency.

Method	Platform	L1 [kB]	L2 [kB]	extRAM [MB]	extFLASH [MB]	Latency [M cycles]	MAC/ cycle
Viola-Jones	GAP8 (CL)	51.6	384	–	–	55.0	–
	GAP9 (CL)	99.6	384	–	–	52.3	–
MBNV3-SSD	GAP8 (CL)	47.6	267	1.6	3.44	255.3	1.5
	GAP9 (NE16)	115.6	1200	1.3	3.44	35.3	10.9

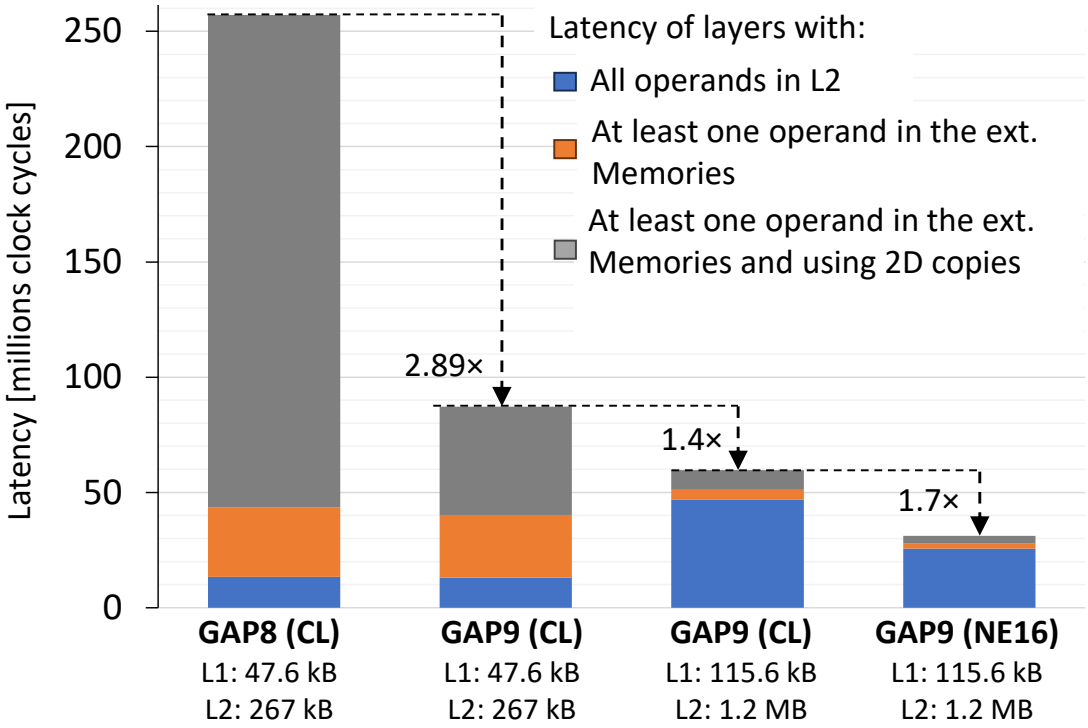

Figure 5.5. Latency analisys of the MobileNetV3-SSDLite execution on GAP8 and GAP9 under varying L1 and L2 memory budgets.

transmission. In the plot, I break down the energy costs for the image sensor, the MCU, namely GAP8 or GAP9 running Viola-Jones or the MBNV3-SSD, and the LoRA module described in Sec. 5.2.3. First, the energy consumption of the camera sensor is negligible with respect to the other system costs. On GAP8, the CNN compute cost is ∼10× higher than the transmission cost (161 mJ vs 17 mJ). I account for the radio's cost to transmit 4 bytes for the pest count and 13 bytes of the LoRaWAN protocol overhead. With Viola-Jones, the energy consumption gap decreases, but the processing (31.8 mJ) still presents the largest energy cost. On the other hand, the energy consumption of the GAP9 SoC, which I measured on an evaluation board comprising the external memories, is 7× and 34× lower

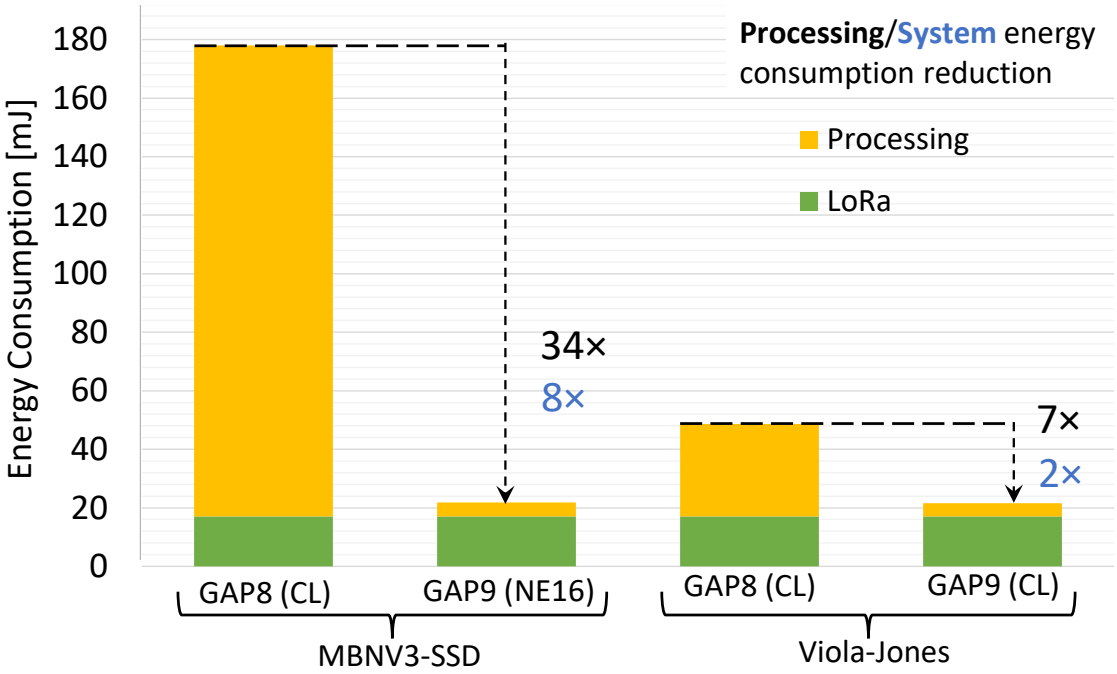

Figure 5.6. Energy consumption of the system during an active cycle composed of camera acquisition, processing, and transmission of the result.

than the energy consumption of the GAP8 MCU for the execution of, respectively, Viola-Jones and MBNV3-SSD. The 59% of this improvement is explained by the micro-architectural innovations of GAP9 vs. GAP8, i.e., more memory and the NE16 accelerator, reducing the clock cycle count.

The rest can be instead attributed to the lower power consumption of GAP9, reaching 20.5 mW and 33 mW for Viola-Jones and the CNN processing, the latter including also the energy consumption of the external memories. On GAP8, I measured a power consumption of 79 mW instead. From a system-level viewpoint, the energy cost of the GAP9 processing unit is approximately $\sim 3.5\times$ lower than the transmission cost, thus enabling energy-efficient near-sensor pest detection with a minimal impact on the overall energy cost. This solution presents a consumption higher of only 0.24 mJ with respect to the lightweight Viola-Jones algorithm while providing a higher accuracy (+1.3% and +27% respectively for the *near-ds* and *far-ds*)

5.3.4 Comparison vs. State-of-the-Art

I evaluate the energy consumption of my method in a complete system composed of a camera, the MCU with its external memories, and a LoRa transmitter as described in Sec. 5.2.3. Tab. 5.3 compares my solution with other SoA camera-based systems for codling moth detection that use MCUs for onboard processing.

The table outlines the computer vision algorithms employed and their processing and transmission costs during the active state. It also details the total daily energy consumption, which accounts for the energy spent in active and deep sleep states, and the energy needed to load the DNN's weights from the external to the on-device memory. I adopt a transmission policy for the system-level evaluation that sends the pest count after detection, acting as a live report within the final application scenario.

The method by *Suto* [20] reaches a detection accuracy of 82%, similar to my solution, despite not being directly comparable because of the diverse dataset (not publicly available). Their MCU, the STM32H7, consumes 30 J of energy for a MobileNetV2 inference, 6300× higher than my best solution executing a similar workload, Their MCU, the STM32H7, consumes 30 J of energy for a MobileNetV2 inference, 6300× higher than my best solution executing a similar workload, leading to a daily consumption 6.9× higher than my CNN and a battery lifetime of only 4.6 days in the low-energy scenario. Furthermore, the platform requires 1 minute to perform the inference, which does not match the requirements of the high-frequency scenario.

In contrast, the accuracy score (93%) reported by [2] only refers to the classification of a set of patches extracted from the frames of my same dataset. However, the background subtraction algorithm, which serves as a patch extractor, can lead to the misidentification of significant patches, thereby compromising the overall accuracy of the pest count. Variations in luminosity can result in the misclassification of these patches, a problem to which CNNs have demonstrated greater robustness [117]. Unlike background subtraction, which relies solely on detecting differences between frames, CNNs do not suffer from the inability to recover from false negatives. If a pest is not detected in the initial frame when it enters the trap, the background subtraction algorithm may subsequently classify it as part of the background, preventing its detection in future frames. In contrast, CNNs, which analyze each frame independently, are not subject to this limitation. For these reasons, the detection accuracy of my method is not directly comparable with the score reported in [2]. A fairer comparison between my methods is to apply their CNN, patch by patch, to the whole image, giving us an energy consumption of 31.5 mJ (6.6× more than ours). Lastly, my object detection algorithm deployed on the GAP9 platform consumes 4.85 mJ of energy. This is only 1.35 mJ more than the energy consumption reported in [2], which used a low-frequency operating point (50 MHz) on the GAP8 platform. However, it is important to note that [2] did not account for the energy cost of the external memories.

Since the system described in [2] transmits image patches upon detection, its daily energy consumption is 1728.7 and 1719.0 J for the high-frequency and low-energy scenarios, respectively. In this estimate, I considered the transmission cost from [2]

and an average of approximately 33 codling moths detected per trap per day, as reported in [118]. To fairly compare this system with my work, I calculated the energy consumption in the case my node sends an image after every detection (a total payload of 12.7 kB that includes a JPEG-compressed image and the overhead of the LoRaWAN protocol). Given that my LoRa module consumes 1 mJ per byte sent, the total energy consumption is 12.7 J per image. For 33 detections per day, as reported in [118], my system's daily energy consumption is 445.0 J in the high-frequency scenario and 423.4 J in the low-energy scenario. This daily energy consumption is approximately 3.9× lower than the system described in [2] under the same conditions. Considering that I send only pest counters with the same frequency of detections, the energy consumption of my system is reduced by 6.5 times compared to sending the frame if a detection is made, achieving a lifetime of 199 days.

GAP9 achieved a 7× reduction in processing energy compared to my previous work [50], which used GAP8 as the main processing unit. This significant reduction in processing cost decreases the daily energy consumption by 76.2 J for high-frequency and 1.9 J for low-energy scenarios. The proposed system can achieve a lifetime of 170 days for MBNV3-SSD and 199 days for Viola-Jones in the high-frequency scenario, and 2257 days for MBNV3-SSD and 2296 days for Viola-Jones in the low-energy scenario when sending pest counts at the same frequency as detections. In both scenarios, this exceeds the expected activity period of Cydia pomonella of approximately 120 days [112].

Table 5.3. Comparison of Pest Detection Camera-based systems with ultra-low power MCUs for onboard processing.

	Method	Processor	Pest Detections Latency	Processing Cost [mJ]	Comm. Module	Daily energy consumtpion [J]			
						High-frequency		Low-energy	
						Transmitting images	Transmitting pest counters	Transmitting images	Transmitting pest counters
J.Suto [20]	DNN (Parameters: 3.2 M) Acc:82%	STM32H7 480 MHz	1 min	30600	GSM images	—	—	2937.6	—
Brunelli et al. [2]	GMM+DNN (Parameters:ND) Acc:93%[a]	GAP8 50 MHz	810 ms + 51 ms/inference	**3.5** [b]	LoRaWAN images	1728.7	—	1719.0	—
Rusci et al. [50]	Viola-Jones (Parameters: 3.6 k) Acc:81.7%	GAP8 175 MHz	400 ms	31.8	LoRaWAN pest counter / images	513.2	143.1	424.8	7.7
This Work	Viola-Jones (Parameters: 3.6 k) Acc:81.7%	GAP9 240 MHz	221 ms	4.61		**437.5**	**66.9**	**423.3**	**5.8**
	MBNV3-SSD (Parameters:3.44 M) Acc:83%	GAP9 (NE16) 240 MHz	147 ms	4.85		445.0	74.4	423.4	5.9

[a] only tested on crops, [b] not accounting for external memories,

Chapter 6

Application Specific Acceleration for Heterogeneous MCUs

In this final chapter, I apply the hardware–software co-design methodology to two workloads, Speech Enhancements (SEs) and VODs , illustrating how the careful alignment of algorithmic structure with architectural capabilities enables efficient execution of complex workloads on ultra-low-power platforms. The proposed methodology emphasizes tailoring computation to the available hardware resources and exploiting the peculiarities of the application itself, balancing numerical precision, energy efficiency, and real-time responsiveness within the constraints of heterogeneous microcontroller-class systems.

For VODs, Sec. 6.1, I preserve floating-point inference on the on-chip FPU and introduce a multi-resolution policy that interleaves full- and low-resolution frames to lower per-frame cost. A Kalman-based tracker exploits temporal continuity to maintain object identities across frames. I also add a probability-based Rescore mechanism that corrects early or sporadic classification errors: class posteriors for each tracked trajectory are updated over time using detector confidences (and their reliability at different resolutions), so that transient misclassifications are down-weighted while consistent evidence accumulates. This probabilistic refinement, coupled with tracking, recovers accuracy lost to low-resolution passes and stabilizes category assignments without additional parameters or retraining.

For multi-channel SEs, Sec. 6.2 via neural beamforming, Sec . 6.2, I split the pipeline by sensitivity to numerical precision. A lightweight convolutional network, run in `int8` on the `NE16` accelerator, produces time–frequency masks that estimate speech and interference components and provide reliable statistics for spatial filtering. The subsequent Minimum Variance Distortionless Responses (MVDRs) beamformer are

executed in `float32` to preserve numerical stability: they estimate spatial covariance matrices from the masked spectra and compute filters that suppress noise while preserving the integrity and directionality of the speech signal. This division of labor is complemented by an interleaved execution strategy, wherein beamforming is performed concurrently with the computation of new weights by the neural network. In this way, the pipeline exploits quantization-tolerant network inference in the mask-estimation stage while dedicating floating-point precision to the covariance inversion and constraint enforcement processes that are numerically delicate, achieving both real-time performance and stable enhancement quality.

6.1 Video Object Detection

A Video Object Detection (VOD) algorithm identifies objects in a video stream by reporting their categories and the corresponding coordinates in the image space. This function is critical for various cyber-physical systems like surveillance cameras [119] and miniaturized (as big as the palm of a hand) drones [120, 121]. These systems typically integrate a camera with an ultra-low-power embedded processor, i.e., a low-cost Microcontroller Unit (MCU). Compared to other edge/mobile processors (e.g., Nvidia Tegra, Raspberry Pi, Qualcomm Snapdragon, etc.), MCUs feature a 10-100× power consumption to enable battery-powered operations (also referred to as the *extreme edge computing* [122]). On the other side, these devices present a limited on-chip memory (typically a few MB) and low computational power [123]. These severe limitations challenge the design of high-throughput VOD systems using compute-intensive accurate vision algorithms.

Recently, VOD was shown on multi-core MCUs [123] using lightweight (i.e., with a low memory budget) Deep Neural Network-based (DNN) object detectors with a simple frame-by-frame analysis [120, 19, 121]. For every frame of the image stream, the DNN inference task returns a list of output detections with confidence scores between 0 and 1. For example, Fig. 6.1 shows the output of the recent NanoDet-Plus object detector [3] featuring 1.17 million parameters when applied over three consecutive video frames. The precision-recall tradeoff is adjusted by thresholding the confidence scores, as shown in the figure: a higher threshold value leads to fewer false positives (higher precision) or undesired miss-detections (lower recall). From a computational cost perspective, a common strategy to reduce the number of operations per frame, and thus increasing the processing throughput, consists of feeding the model with low-resolution (low-res) input images [124, 63], e.g., -296 M Multiply-Accumulate (MAC) using 2.8 × smaller images in my example.

To tackle this problem, this thesis proposes *Multi-Resolution Rescored ByteTrack* (*MR2-ByteTrack*), a method that transforms an image object detection model into a full VOD algorithmic pipeline, simultaneously improving the detection accuracy and reducing the overall computational cost with respect to a naïve *frame-by-frame*

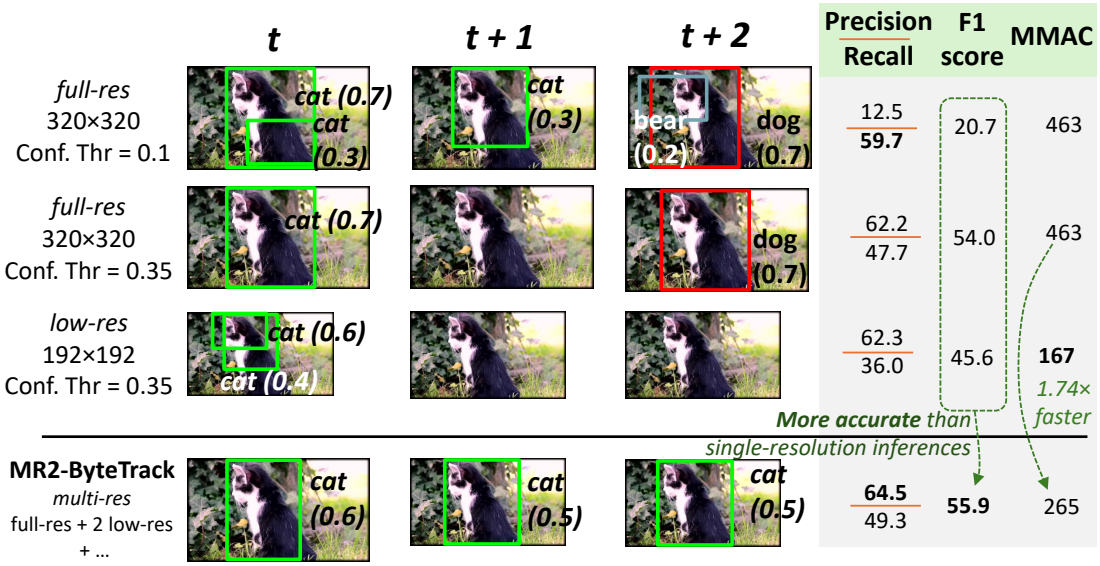

Figure 6.1. Video object detection using NanoDet-Plus [3] object detector under multiple thresholds and input size settings vs. the proposed *MR2-ByteTrack* solution.

processing pipeline. MR2-ByteTrack combines an off-the-shelf DNN object detector with an extended version of *ByteTrack* [125]. This lightweight Kalman-filter-based tracker updates the output detections based on the object instances already detected by the DNN. Because the original ByteTrack [125] lacks recovery mechanisms from misclassification (e.g., a cat is tracked as a "dog" if wrongly predicted at the beginning of the video sequence), I propose a novel *Rescore* algorithm to refine classifications over time, according to a probabilistic logic.

My approach adopts a *Multi Resolution Inference* scheme to reduce the per-frame workload, inserting low-resolution (low-res) frames between full-resolution (full-res). As shown at the bottom of Fig. 6.1, my MR2-ByteTrack leverages the temporal correlation of detections in consecutive frames to recover the accuracy drop of low-resolution inferences. When deployed on a multi-core MCU, this method incurs no additional memory cost for parameter storage compared to a single-resolution inference scheme, as the same object detection DNN is applied to both full and low-resolution frames.

In summary, this chapter makes the following novel contributions:

- The MR2-ByteTrack framework, combining an off-the-shelf DNN-based object detector for multi-resolution inference, the ByteTrack Kalman-based tracker, and the *Rescore* method to refine category assignment of tracked frames, reducing misdetections and misclassifications;

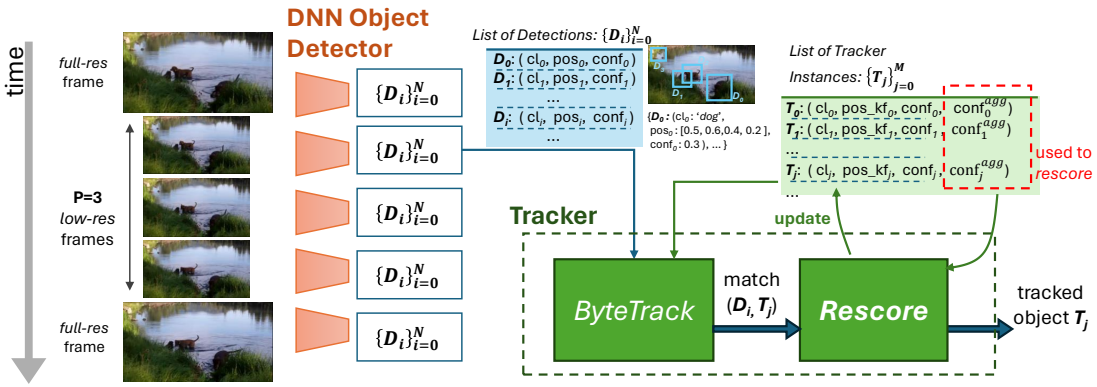

Figure 6.2. Overview of the proposed Multi-Resolution Rescaled ByteTrack algorithm for video object detection.

- The first embodiment of a multi-resolution VOD pipeline on an ultra-low-power MCU, Greenwaves Technologies (GWT)'s GAP9, showing a better accuracy-throughput tradeoff than single-resolution inference VOD systems.

I validate my proposed MR2-ByteTrack method using three State-of-the-Art (SoA) tiny object detectors: YOLOX-Nano [28], NanoDet-Plus [3], and EfficientDet-D0 [27], all off-the-shelf pretrained on the COCO dataset [26]. Across the ImageNetVid [126] validation set, MR2-ByteTrack improves mean Average Precision (mAP) scores compared to the object detector baselines by up to 5.17% and the F1 score by 3.58% if applied only on full-res frames. If, instead, I interleave for each full-res frame two low-res frames, the mAP improvement is +2.16% while the average MAC computational costs are reduced by up to 43%.

I deployed my multi-resolution VOD strategy, using the on the GAP9 MCU processor featuring a 9-core compute cluster at a peak power cost of up to 72 mW, also considering the external 32 MB and 8 MB MB FLASH and RAM. Compared to the baseline object detectors [28, 3, 27], MR2-ByteTrack achieves up to 1.76× lower inference latency with no increase in the DNN's parameters footprint (2.24 MB for the largest model) and only a modest increase in code size (+186 kB), thus enabling high-accuracy VOD on milliwatt-power extreme-edge devices. Also, when compared to the SoA YOLOV method [62], my MR2-ByteTrack shows the same F1 score but saves the memory and computes overheads (14% and 21%) of the transformer-based front-end that differently, from my ByteTrack module, demands an ad-hoc training process.

6.1.1 Method

Multi-resolution Rescored ByteTrack

The MR2-ByteTrack algorithm in Fig. 6.2 combines three different components: (*i*) a vision-based Convolutional Neural Network (CNN) object detector that takes in input images of different sizes; (*ii*) the ByteTrack tracking algorithm [125]; (*iii*) the novel *Rescore* algorithm to update the detection scores returned by ByteTrack.

Multi-resolution Object Detector Backbone

I design my framework around existing pre-trained single-shot object detectors. In my formulation, the detection algorithm includes a lightweight CNN model, i.e., the backbone, to analyze the image frames of a video sequence in real-time. Given a frame F at time t, I indicate the set of detections the processing task returns as $\{D_i\}_{i=1}^{N}$, where N is the total number of objects detected within the current frame. Each detection D_i is described by a triplet $(\text{bbox}_i, \text{cl}_i, \text{conf}_i)$: a bounding box bbox, a class index cl and a confidence score conf < 1. Detections with a confidence score lower than a *low_threshold* are discarded, while the others are retained for further processing.

Since I employ a CNN-based backbone, I can interleave one full-res image (e.g., 320×320 px in my setup) with P low-res ones (e.g., 192×192 px). This way, I can trade detection accuracy for fewer MAC operations. Thus, the average number of MAC operations per frame becomes:

$$\text{MAC} = \rho \cdot \text{MAC}^{fr} + (1 - \rho) \cdot \text{MAC}^{lr} \tag{6.1}$$

Where $\rho = \frac{1}{1+P}$, MAC^{fr} and MAC^{lr} represent the number of MAC operations required to execute the inference on one full-res and one low-res frame, respectively. Because the inference task *utilizes the same network parameters* in both low and full-resolution cases, the weight footprint remains the same as with single-sized frames. In fact, the convolution layers of the CNN models can apply to any scale of input images or feature maps. As an example, a 3x3 filter can slide over a larger feature map to produce a wider output tensor, corresponding, eventually, to a higher number of detections.

ByteTrack Tracker

Naïve frame-by-frame object detectors can not exploit any temporal correlation between frames, i.e., detections in consecutive frames are likely to stay the same or similar. I use the ByteTrack [125] tracker to exploit this correlation, recovering for performance losses due to down-scaling (i.e., low-res images) and possibly improving the precision and recall of a frame-by-frame baseline (see Sec. 6.1.5).

The backbone's detections $\{D_i\}_{i=1}^{N}$ are fed to the ByteTrack algorithm based on a Kalman filter for every frame. In its original formulation, at time-step t, ByteTrack operates on a list of M instances: $\{T_j\}_{j=1}^{M}$, i.e., one instance for each tracked object. Like for detections, each tracker instance (T_j) is described by a triplet $(\text{bbox\_kf}_j, \text{cl}_j, \text{conf}_j)$. bbox_kf_j is the bounding box predicted by a Kalman filter fed with the backbone's detections. cl_j and conf_j are the class index and confidence scores associated with tracked objects, respectively.

To correlate the detection performed over time, ByteTrack matches the bbox bounding boxes of $\{D_i\}_{i=1}^{N}$ and the predicted bbox_kf ones of $\{T_j\}_{j=1}^{M}$, using a cost matrix determined by Intersection-over-Union (IoU) scores. An IoU threshold of 0.3, as proposed in the original paper, sets the minimum value to determine a match between a detection and a tracker instance. Unlike the original implementation, I do not adopt any feature-matching criteria to associate tracker instances and detections: the spatial size of the feature maps to match changes over time in my multi-resolution framework. A tracker instance is marked active after a minimum number of consecutive detections has occurred (two in my setting). At the same time, it is removed from the tracker list if it was never updated during the last five time steps. After matching, the detection bbox updates ByteTrack's Kalman filter. The remaining detections not matched with any existing tracker instance produce a new one if their conf score is higher than a tuned *high_threshold*.

In the original ByteTrack algorithm, the class index cl_j and the confidence score conf_j of active tracker instances are never updated after the match with a new detection. I chose to modify this logic to better cope with the multi-class nature of VOD: specifically, I update cl_j and conf_j if the confidence score of the matched detection is higher than the *high_threshold* value.

Rescore Algorithm

I augment ByteTrack with the *Rescore* algorithm to improve the estimation of the class index and confidence scores of an active tracker instance by accounting for the history of matched detections. To this aim, I extend the tracker instance T_j with a class status attribute $\text{conf}_\text{j}^{\text{agg}}$, which models the probability of the tracker's class cl to be correct. The $\text{conf}_\text{j}^{\text{agg}}$ value aggregates the confidence scores conf of the detections assigned to the j-th tracker instance until time $t-1$. During this aggregation, the conf value is interpreted as the probability that the object's predicted class is correct in the current-frame detection.

When a new detection D_i is assigned to the tracker instance T_j at time t, the class index cl_j is *rescored* based on $\text{conf}_\text{j}^{\text{agg}}$ as shown in Algo. 1. If the class index cl_i is the same as cl_j, I only update the $\text{conf}_\text{j}^{\text{agg}}$ value, as the match confirms my confidence in the correctness of the classification. I use the product of the opposites of $\text{conf}_\text{j}^{\text{agg}}$ and conf_i as the new value for the status variable. If the class indices do not

Algorithm 1: Rescore algorithm

Inputs: Match $\{D_i = (\text{cl}_i, \text{conf}_i), T_j = (\text{cl}_j, \text{conf}_j, \text{conf}_\text{j}^\text{agg})\}$
if $\text{cl}_j == \text{cl}_\text{i}$ **then**
  $\text{conf}_\text{j}^\text{agg} \leftarrow 1 - ((1 - \text{conf}_i) * (1 - \text{conf}_\text{j}^\text{agg}))$
else
  if $\text{conf}_\text{j}^\text{agg} < \text{conf}_i$ **then**
    $(\text{cl}_j, \text{conf}_j, \text{conf}^\text{agg}{}_j) \leftarrow (\text{cl}_i, \text{conf}_i, \text{conf}_i)$
  else
    $\text{conf}_\text{j}^\text{agg} \leftarrow 1 - ((1 - \text{conf}_\text{j}^\text{agg})/(1 - \text{conf}_i))$
    $\text{conf}_\text{j}^\text{agg} \leftarrow max(\text{conf}_\text{j}^\text{agg}, 0)$
    if $\text{conf}_\text{j}^\text{agg} < \text{conf}_i$ **then**
      $(\text{cl}_j, \text{conf}_j, \text{conf}_\text{j}^\text{agg}) \leftarrow (\text{cl}_i, \text{conf}_i, \text{conf}_i)$
    end
  end
end
$\text{conf}_\text{j}^\text{agg} \leftarrow min(\text{conf}_\text{j}^\text{agg}, 1 - \epsilon)$
return $T_j = (\text{cl}_j, \text{conf}_j, \text{conf}_\text{j}^\text{agg})$

match, I acknowledge this by decreasing my aggregate confidence, i.e., I decrease the $\text{conf}_\text{j}^\text{agg}$ value (the max guarantees a value >0). Then, I check for a class index rescore: if the new detection's confidence is higher than the $\text{conf}_\text{j}^\text{agg}$, I update the tracker class with the new detection: $\text{cl}_\text{j} \leftarrow \text{cl}_\text{i}$. Additionally, I impose an upper bound (i.e., $1 - \epsilon$) to the $\text{conf}_\text{j}^\text{agg}$ to prevent overconfident detections from blocking future rescores. Finally, the *Rescore* algorithm also outputs the mean confidence score during the tracking.

6.1.2 Experimental Results

Multi-resolution Inference on MCUs

I target ultra-low-power embedded systems, such as tiny MCUs, to deploy my pipeline. Since the end-to-end latency of VOD applications is dominated by DNN inference, I extensively benefit from the throughput optimization of Eq. 6.1. To demonstrate my approach's real-time performance and power consumption, I select a GWT GAP9 MCU, see section 3.2 for a description of the platform. In this chapter, I do not use the on-board NE16 accelerator, which would require lossy 8-bit quantization, and instead exploit the floating-point multi-core support of the platform to avoid any accuracy degradation.

I use GWT's GAP*flow* toolset to generate the target platform's inference `C` code. For every trained model with a specific input size, the tool generates header files containing the model parameters and a source file with graph-level and layer-level routines. Following the automatically generated memory management scheme, weight parameters are stored within an external octoSPI Flash memory. At the same time,

Table 6.1. Baseline Object Detection Models

Model	Params	full-res (px)	MMAC^{fr}	low-res (px)	MMAC^{lr}
YOLOX-Nano	0.9 M	320×320	316	192×192	114
NanoDet–Plus	1.18 M	320×320	463	192×192	167
EfficientDet-D0	3.93 M	384×384	1440	256×256	640

Table 6.2. Impact of the Rescore Algorithm on ByteTrack using only high-res frames ($P = 0$).

Method	**mAP**	**Precision**	**Recall**	**F1 score**
YOLOX-Nano	41.30	**65.16**	44.92	53.18
w/ Naïve-ByteTrack	43.57	63.9	47.96	54.79
w/ MR2-ByteTrack	**46.47**	64.24	**51.23**	**57.00**
NanoDet–Plus	42.70	62.15	47.68	54.00
w/ Naïve-ByteTrack	44.00	58.91	51.40	54.90
w/ MR2-ByteTrack	**46.57**	**64.14**	**51.50**	**57.10**
EfficientDet-D0	60.70	78.48	64.14	70.30
w/ Naïve-ByteTrack	61.09	78.65	66.42	72.02
w/ MR2-ByteTrack	**64.86**	**80.63**	**67.32**	**72.52**

the intermediate values of the inference tasks, i.e., the activation tensors, are allocated within the on-chip L2 memory and the external octoSPI RAM. To cope with my multi-resolution scheme, I generate two `C` implementations, one for the low-res case and a second for the full-res one. The two network functions access the same weight data stored in the Flash memory.

On GAP9, the latency cost of downscaling the input image is negligible compared to the inference time. The overhead for generating a low-res image of 192×192 px from a 320×320 px image is 1.5 ms. This corresponds to only 0.9% of the inference time in the case of NanoDet-Plus (169 ms). The Kalman filter also has low execution time and requires only 9.5 kB of on-chip memory. While my implementation of the MR2-ByteTrack method is specific to the GAP9, I remark on the generality of the method that can be easily applied to other MCUs.

6.1.3 Baseline Models and Metrics

I consider three State-of-the-Art object detectors trained on the COCO dataset: YOLOX-Nano, NanoDet-Plus, and EfficientDet-D0. Tab. 6.1 reports the number of

Table 6.3. VOD solutions on the GAP9 MCU: Mr2-ByteTrack vs. frame-by-frame CNN object detectors with full-res or low-res inputs.

Method	Input Size	RAM [MB]	Flash Mem. [MB]	Code [kB]	[FPS]	Energy [mJ]	mAP	Precision	Recall	F1 score
NanoDet–Plus	*single* full-res	1.6	2.3	186.0	3.3	21.8	42.7	62.2	47.7	54.0
NanoDet–Plus	*single* low-res	0.6	2.3	186.0	**9.8**	**7.5**	27.4	**62.9**	31.3	41.8
w/ MR2-ByteTrack (mine)	*multi-res* P=2	1.6	2.3	373.0	5.9	12.3	**44.9**	61.5	**49.3**	**54.7**
YOLOX-Nano	*single* full-res	1.4	1.8	103.0	3.3	19.2	41.3	**65.2**	44.9	53.2
YOLOX-Nano	*single* low-res	0.5	1.8	103.0	**8.6**	**7.0**	25.7	63.2	28.8	39.6
w/ MR2-ByteTrack (mine)	*multi-res* P=2	1.4	1.8	205.0	5.5	11.1	**41.4**	64.0	**45.7**	**53.3**

parameters and MMAC operations for the input frames' considered full-res and low-res sizes. For VOD experiments, I use the ImageNetVID dataset, which shares 16 classes with the COCO dataset. By extracting video sequences that share common classes between the two datasets, I build my validation set of 392 video samples. To distinguish from the entire dataset, I indicate this subset as ImageNetVID^C. To assess the detection performance, I use the mAP50 score, a widely used metric in literature, which I call mAP in the rest of the paper. In addition, I report the precision, recall, and F1 score averaged over the classes. Differently from mAP, which expresses the mean value of the per-class average-precision curves, these three additional metrics show the trade-off between true detections, false positives, and miss-detections. All metrics consider an IoU of 0.5 to mark a correct match between ground truths and predictions.

6.1.4 Single-resolution Trackers

Tab. 6.2 compares the results obtained with my MR2-ByteTrack against the baseline object detection models and the Naïve-ByteTrack on the ImageNetVid^C dataset. In this initial experiment, I feed the inference models only with full-res images. The confidence threshold of the baseline models is set to 0.35, 0.3, and 0.4 for, respectively, NanoDet-Plus, YOLOX-Nano, and EfficientDet-D0. These values are optimally tuned to obtain the highest F1 score on the testing dataset and reflected on the *high_threshold* value of the ByteTrackers. Using the same procedure, I also set the *low_threshold* of the trackers to 0.3, 0.25, 0.35 for NanoDet-Plus, YOLOX-Nano, and EfficientDet-D0, respectively. By observing the results, I can see that, on average, the Naïve-ByteTrack increases the mAP score measured on the baseline models by 1.55 %. This is explained by the capacity of the tracker to positively aggregate low-confident detections, which the baseline models instead discard.

Next, the Rescore algorithm of MR2-ByteTrack leads to a further 3.08% improvement by turning false-positive detection into correct classification, as can be observed by the higher Precision, Recall, and F1 scores. Overall, thanks to my MR2-ByteTrack, the smallest NanoDet-Plus and YOLOX-Nano models can achieve up to 46.57 and 46.47 mAP on high-res frames, respectively, +3.9 % and +5.2 % vs.

Table 6.4. Comparison between VOD methods on ImagenetNetVID.

Method	Object detector Backbone	Prec.	Recall	F1 score	mAP	Params [M]	GMAC	vs. full-res object detetector		
								Δ mAP	Δ params	Δ MAC
YOLOV-S [62]	YOLOX-S[28]	79.9	66.4	72.5	62.5	9.9	12.9	2.7	+14.0%	+21.0%
MR2-ByteTrack		80.8	65.9	72.5	61.8	9.0	10.9	2.0	0	0
MR2-ByteTrack (P=2)		77.5	64.3	70.3	60.3	9.0	6.2	0.5	0	-43.0%
Liu et. al [65]	SSDLite-Mobilenetv2[25]	n/a	n/a	n/a	61.4	4.9	0.2	0.9[1]	+11.0 % [1]	-84.0%
MR2-ByteTrack (P=2)	EfficientDet-D0[3] [27]	80.0	64.1	71.1	61.2[2]	3.9	0.9	0.5[2]	0	-37.0%

[1] w.r.t. large f_0 non-interleaved model [65], [2] measured on ImageNetVID^C, [3] trained on the COCO dataset,

the baselines.

6.1.5 Multi-resolution Performance

Fig. 6.3 plots the measured mAP of MR2-ByteTrack when varying the amount of interleaved low-res images (P from 0 to 10) in the multi-resolution setup, and I benchmark against the baseline models in the same setup. The bars on the bottom show the average GMAC operations required by each configuration (Eq. 6.1). For baseline models, the mAP starts dropping at P=1, while MR2-ByteTrack decreases performance with $P \geq 5$ due to low-res images. This difference between methods arises from the greater robustness of my MR2-ByteTrack approach. This effect is related to the inertia of the trackers, which are kept alive for up to 5 timestamps without new detections in my setup.

Overall, the mAP scores of the MR2-ByteTrack models using multiple low-res images are always better than the baseline models operating on full-resolution frames. For P=1 and P=2, on average, my approach marks a mAP improvement of 3 % and 0.96 %, while the number of MAC operations is reduced by 32 % and 43 %, respectively, compared with the baseline object detectors. For $P \geq 3$, I observe a progressive reduction of mAP below the original object detection performance, which can be affordable for some use cases. Finally, I chose the configuration P=2 for the remainder of my experiments as a good trade-off between accuracy and computational load (i.e., MAC operations).

6.1.6 State of the Art Comparison

MCU-ready VOD Systems

Tab. 6.3 compares my MR2-ByteTrack solution with P=2 vs. frame-by-frame single-resolution baselines (e.g., used by [19, 120]) with full-res and low-res inputs when running on an MCU system. To this aim, I deploy the different approaches on the GAP9 SoC, coupled with two external memories: a Flash of 64 MB and a RAM of 8 MB. The table reports the code size, the activation and weight memory

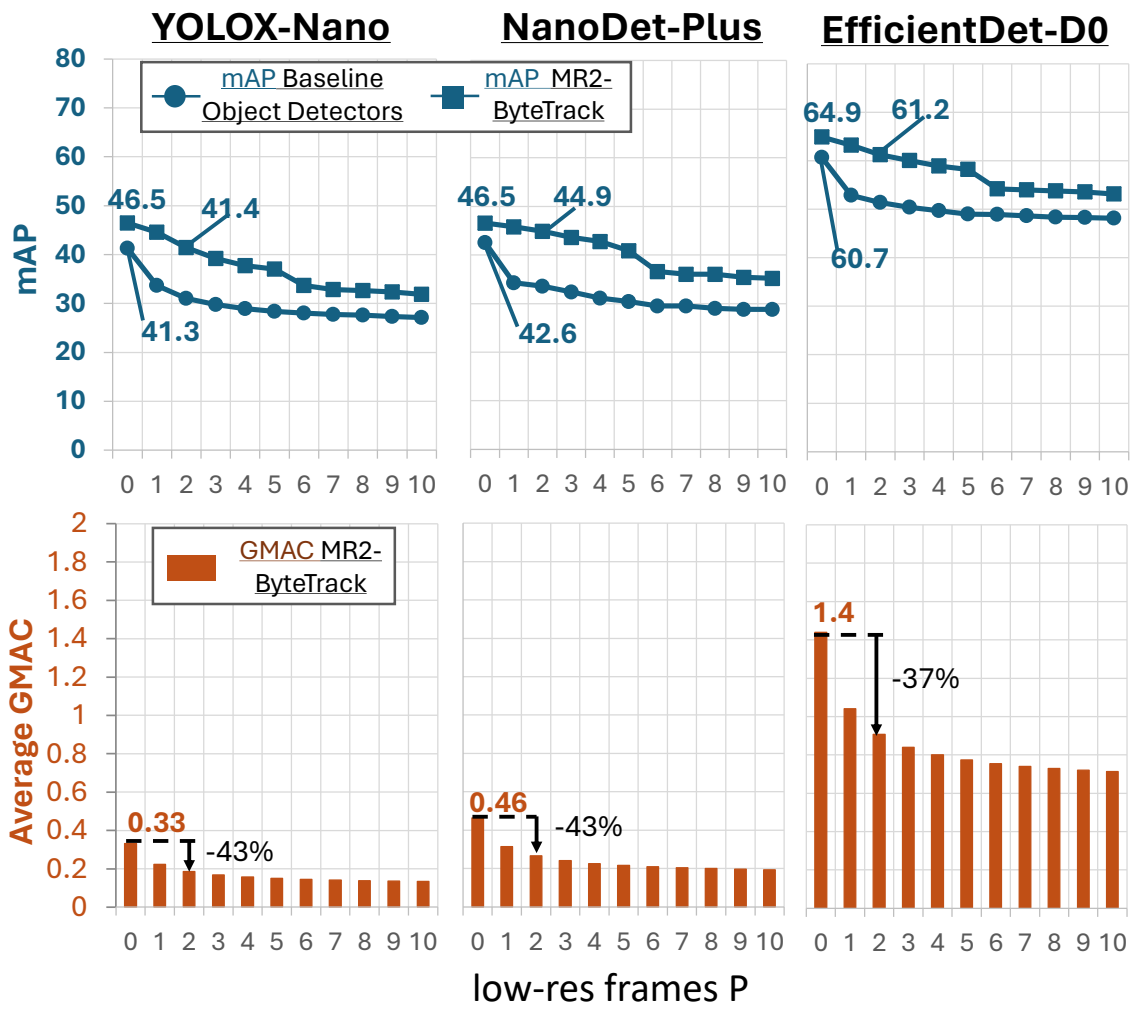

Figure 6.3. mAP (blue) and GMAC (red) of MR2-ByteTrack at varying low-res frames vs. the baseline.

occupation in MB, denoted as RAM and FLASH memory, the processing throughput in FPS, and the energy cost, measured when the MCU is clocked at 370 MHz, together with the detection metrics.

I show results for YOLOX-Nano and NanoDet-Plus, while I leave out of this analysis the EfficientDet, as its memory footprint is larger than the available budget. The datatype of weights and activations is cast to half-precision `FP16` for a lossless deployment. On GAP9, my multi-resolution instance features a double code size (up to 373 kB in total) vs. baseline models. The activation memory of the high-res model dominates the total cost of the multi-res deployment; the weight storage is constant among different resolutions. My solutions can run at 5.9 FPS

with NanoDet-Plus and 5.5 FPS with YOLOX-Nano, 1.7× faster than the high-res version. On the other side, the cost of the ByteTrack algorithm is negligible (<0.2% than the inference). The latency gains are reflected in the energy costs: the multi-resolution models consume 1.77× and 1.72× less than the high-res versions for, respectively, NanoDet-Plus and YOLOX-Nano. Thus, observing the highest mAP and F1 scores, *my MR2-ByteTraker shows the best energy-accuracy trade-off for VOD applications on MCU systems.*

VOD Algorithms

Tab. 6.4 compares my approach with recent real-time methods *not* tailored for MCU deployment given the high number of parameters of the DNN backbones: 9 M for YOLOX-S in the YOLOV work [62] and 4.9 M for SSDLite-MobileNetV2 in the study by *Liu et al.* [65]. To fairly compare vs. YOLOV, which uses a transformer to aggregate the temporal detections of the YOLOX-S backbone, I apply my MR2-ByteTrack to the same CNN object detector pre-trained on the ImageNetVid trainset, considering both the high-res or the multi-resolution (P=2) scenarios. For YOLOV, I test the real-time version in which only the detections of the past frames are sent to the transformers. The scores reported in the table adopt confidence threshold values that maximize the F1 score.

My MR2-ByteTrack reaches a +0.9 higher precision than the YOLOV due to the lower number of false positives. At the same time, the lower recall (-0.5) acknowledges the recognition ability of the transformer layer, which increases the memory and computational cost by, respectively, 14% and 21% compared to the baseline backbones. My method presents the same F1 score as the real-time YOLOV, saving the transformer overhead costs. Additionally, with P=2, the F1 score is reduced only by 2.2% with a computational cost reduction of up to 43%.

On the other side, because the pre-trained SSDLite-Mobilenetv2 model adopted by *Liu et al.* [65] is not openly available, I take the EfficientDet-D0 model pre-trained on COCO for comparison purposes (EfficientDet-D0 features 1 M parameters less than SSDLite-Mobilenetv2). My MR2-ByteTrack method achieves a substantial latency improvement (37% vs. 84% of [65] that uses P=10) while slightly increasing the mAP score compared to the baseline. However, my solution does not increase the memory footprint (+11% in [65]) and does not require any training of the tracker on the video sequence, highlighting the higher flexibility of my approach.

6.2 Minimum Variance Distortionless Response

SEs is a critical component in hearable devices, as it not only improves user communication by attenuating environmental noise but also enhances the reliability of interactions with voice-activated AI assistants. Most existing approaches, however,

operate on single-channel inputs, as multi-channel processing imposes significantly higher computational and memory demands [127, 128, 129, 130]. Despite this cost, SE methods that use an array of microphones, such as neural beamforming [131], called multi-channel methods, offer clear advantages: they provide superior suppression of background noise and, importantly, preserve the spatial cues of the acoustic scene, thereby maintaining the directionality of the target signal [132]. These properties are particularly relevant for hearable devices, where maintaining intelligibility and localization is critical to user experience.

At the same time, hearable devices are strongly constrained by the limited size and battery capacity, along with the need for all-day operation and real-time execution. These constraints can only be met by microcontroller-class processors, which typically provide just a few MB of on-chip memory. As a result, mapping multi-channel deep learning (DL)-based SE algorithms onto such platforms requires careful design of algorithms and optimized software implementations.

In this chapter, I present a deployed multi-channel neural beamforming pipeline on a microcontroller-class device for usage on hearable devices. My method combines a convolutional neural network for mask estimation with a minimum variance distortionless response (MVDR) beamformer, yielding a multi-microphone, beamforming-based SE system. To satisfy the strict requirements of real-time speech processing in hearable devices, typically below 20 ms [133], I introduce an interleaved execution strategy that overlaps beamforming weights estimation with their application. I further employ mixed-precision quantization, executing the neural network in `int8` while retaining the MVDR stage in `float32` for stability, and integrate an efficient speech activity detection (SAD) module to suppress unnecessary computation during silence.

I evaluate my pipeline on GAP9, an ultra-low-power RISC-V-based System-on-Chip (SoC), considering both algorithmic accuracy and hardware deployment metrics. The results demonstrate that my approach achieves real-time operation while preserving enhancement quality and extends the feasibility of advanced multi-channel SE algorithms in wearable devices.

My main contributions are summarized as follows:

- I introduce an interleaved execution strategy that overlaps beamforming and weight estimation to satisfy the 20 ms latency constraint of real-time execution.
- I apply mixed-precision quantization to balance efficiency and accuracy, executing the neural network in `int8` while preserving MVDR beamforming in `float32`.
- I integrate a lightweight SAD module that prevents unnecessary computation during speech absence, improving energy efficiency.

- I deploy a multi-channel neural beamforming pipeline on the GAP9 ultra-low-power SoC, achieving real-time operation for hearable devices.

From my evaluation, the proposed system achieves a short-time objective intelligibility (STOI) score of 0.88 while requiring 17.28 mJ per beamforming inference and 0.62 mJ per SAD inference. Considering a tiny 100 mAh, 3.7 V battery (appropriate for a smart hearable use-case), this corresponds to approximately 16 hours of continuous operation under realistic conditions, assuming speech absence for 70% of the time [134] and including the energy consumption of six high-end microphones. To the best of my knowledge, this represents the first instance of a multi-channel SE neural beamforming pipeline working in real time, deployed directly on an MCU, demonstrating the feasibility of such systems for hearable devices. I release my work as open-source at the following address: `https://github.com/Anon95B/GAP9Beamforming.git`

6.2.1 Methods

In the following, I describe the structure of my neural beamforming algorithm and the voice activity detection network that I employ in my application. The neural beamforming architecture is reproduced directly following the formulation introduced by Ceolini *et al.* [135].

Signal model and neural beamforming

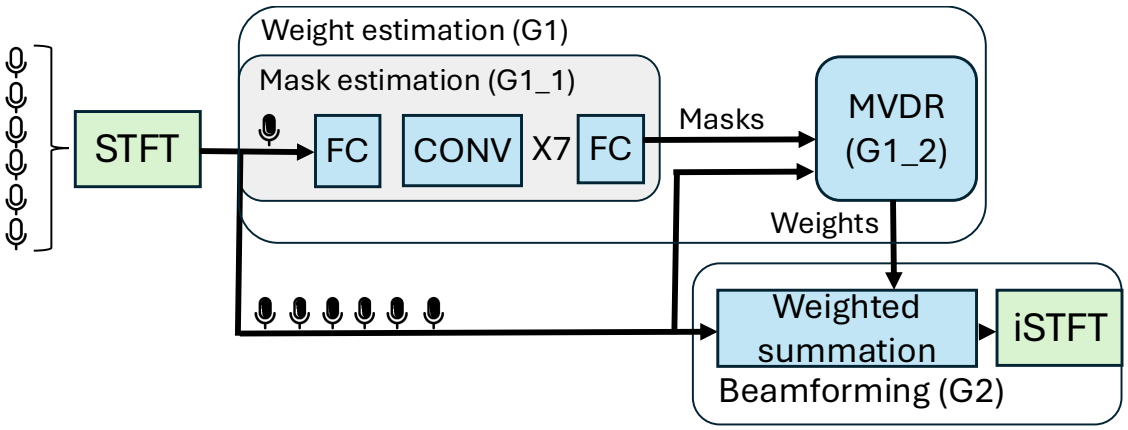

Figure 6.4. Block diagram view of the elements in my neural beamforming application.

I consider a single target speech source captured in noise by M microphones. The signal at microphone m is modeled as

$$y_m(t) = s(t) * r_m(t) + k_m(t), \tag{6.2}$$

where $s(t)$ is the clean speech, $r_m(t)$ the room impulse response, and $k_m(t)$ the additive noise and $*$ indicates the convolution. I assume the narrowband approximation to be valid in my case. After applying the short-time Fourier transform (STFT) [136], the observation becomes a complex signal

$$Y_m(\omega, n) = S(\omega, n)R_m(\omega, n) + K_m(\omega, n), \tag{6.3}$$

with ω denoting the frequency bin and n the frame index. In the following sections, I will refer to the STFT window size as t_{stft} (i.e., the amount of audio each Fourier frame covers) and the hop size as t_{hop} (i.e., the stride between consecutive frames), expressed in milliseconds of audio.

I perform speech enhancement using the minimum variance distortionless response (MVDR) beamformer, which minimizes the noise energy while preserving the target signal. Instead of relying on an explicit steering vector, the MVDR weights are estimated, following the approach proposed in [137], as

$$w_{\text{MVDR}} = \frac{\Phi_{nn}^{-1}\Phi_{ss}}{tr(\Phi_{nn}^{-1}\Phi_{ss})} \tag{6.4}$$

where Φ_{nn} and Φ_{ss} denote the spatial covariance matrices of noise and speech, respectively and tr the trace.

To obtain covariances, I employ a compact convolutional neural network (CNN), referred to as MUDV4 C4 512 in [135], which predicts a time–frequency mask from the spectrogram of a single reference microphone, as illustrated in Figure 6.4. Since spectrograms are complex-valued, I decompose them into real and imaginary parts and concatenate the two as a two-channel input tensor, enabling their use with real-valued neural networks. The CNN consists of a fully connected layer that expands the input dimensionality, followed by a stack of dilated 2D convolutions with residual connections, PReLU activations, and normalization layers. A final projection layer maps the features to a single-channel speech mask $M_{ss} \in \mathbb{R}^{F\times N}$. The corresponding noise mask is obtained as $M_{nn} = 1 - M_{ss}$. Avoiding a separate output head for the noise mask, together with the small number of convolutional features, reduces the overall parameter count to only 20k.

The estimated masks are then used to compute the spatial covariance matrices as

$$\Phi_{vv} = \sum_{n=1}^{N} M_v(n)\, Y(n)Y(n)^H, \quad v \in \{s, n\}, \tag{6.5}$$

which in turn yield the MVDR beamforming weights from Equation 6.4. Applying the MVDR weights to each STFT frame produces the enhanced spectrum, which is then brought back to the time domain using an inverse STFT. The final signal is reconstructed with the standard overlap-and-add procedure, where consecutive

frames are combined to form a continuous waveform. In practice, this means that beamforming plus inverse STFT requires only two adjacent frames, so each new output segment can be generated as soon as t_{hop} of new audio is available.

Interleaved execution strategy

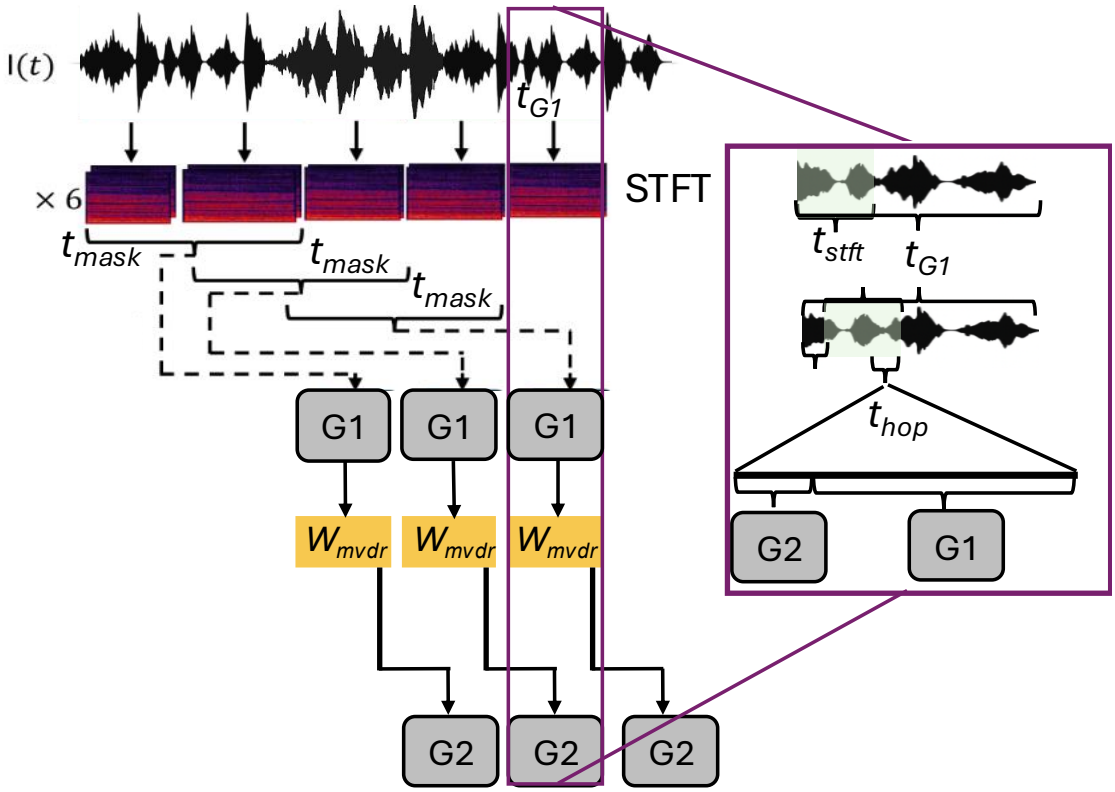

Figure 6.5. Execution schedule of my proposed neural beamforming pipeline.

I propose an *interleaved execution strategy* that partitions the processing pipeline into two computational blocks. As illustrated in Figure 6.4, after the STFT on the input microphones, taking 7 millions of multiply-accumulate operations I have the first block (**G1**) encompasses the mask estimation stage (**G1_1**) followed by the MVDR weight computation (**G1_2**), for a total of 349 million multiply-accumulate (MAC) operations (7 million for the STFT over the six input signals 342 million for the G1_1 and 7 million for the G1_2). The second block (**G2**) then applies the beamforming weights through weighted summation and subsequently performs the inverse STFT to reconstruct the enhanced signal, requiring a total of 781 k MACs (654 thousand for the Weighted summation, 127 thousands for the STFT and its inverse over t_{hop} six input channels and one output channel).

As illustrated in Figure 6.5, execution begins by buffering the required number of

audio samples (t_{mask}) to compute the initial set of beamforming weights. Once the first set of weights has been obtained, I call the time necessary for this computation t_{G1}, the system transitions into the interleaved regime: at each hop (t_{hop}), the current weights are applied to perform beamforming along with the inverse STFT (G2), while the remaining computational budget within the same hop is used to advance G1. In this manner, G1 progresses incrementally, typically one neural network layer at a time, until the next set of weights is ready. Crucially, although the full computation of new weights requires t_{G1}, the execution of the G2 continues without interruption during this period, ensuring that enhanced outputs are generated in real time while weight estimation proceeds concurrently in the background.

Speech Activity Detection (SAD) network

In the absence of speech, estimating speech–noise masks is an ill-defined task; to avoid executing it, I introduce a lightweight SAD network to skip mask estimation and beamforming entirely when no speech is detected. The network is a 1-D ResNet-8 [138] operating on the raw waveform prior to the STFT, composed of an initial convolution, three residual blocks (the first without a pointwise convolution in the skip path), ReLU activations, a global max-pooling layer, and a fully connected head yielding a binary speech/non-speech decision. Although it has 7.6k more parameters than the G1 mask network (27.6k in total), it is far cheaper to run, requiring 4.6× fewer operations than the full speech-enhancement pipeline (76 MMACs), also its structure has been tuned to be efficiently executed on my platform, thereby avoiding substantial latency increase during normal operation.

Hardware platform & deployment

I deploy my neural beamforming application on the GAP9 SoC from GreenWaves Technologies, see Chapter 3.2. The MCus is connected with two external memories (32 MB RAM and 64 MB FLASH for network parameters and activation storage.

I deploy my networks on the device using the GapFlow deployment flow. As the beamforming pipeline requires six input spectrograms to work, it would consume a total of 800 kB of memory to keep everything stored in L2, which would impede the execution of the G1_1. The six spectrograms are only needed from the MVDR step onward, so I keep the spectrogram in the larger external RAM. To maximize the efficiency of my most critical component, the G2, I keep in the faster L2 memory a buffer containing data for each of the six microphones, each storing up to t_{stft} of audio for the G2 block.

6.2.2 Experimental Results

I evaluate the proposed neural beamforming pipeline on GAP9 in terms of memory, quantization, performance, and energy efficiency, and compare it with state-of-the-art speech enhancement systems deployed on MCU.

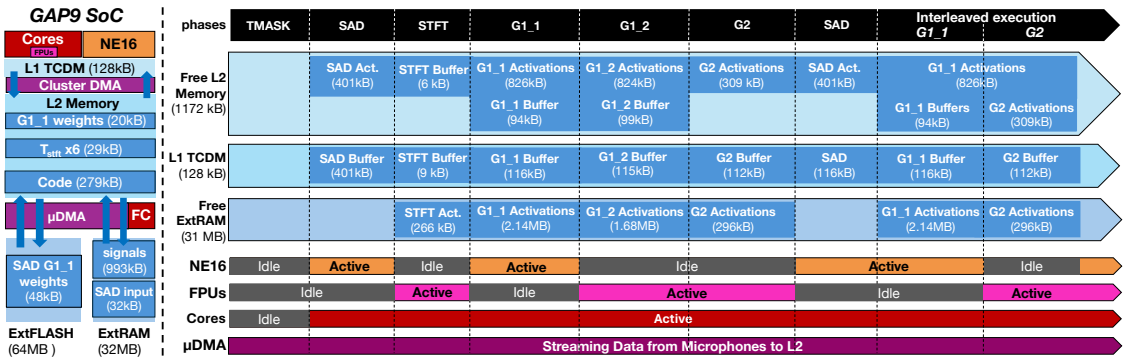

Figure 6.6. (left) Buffer memory allocation and data flow in GAP9 SoC. (right) Beamforming pipeline resource allocation and scheduling in the various execution phases.

Setup & model training

I evaluate my approach using a simulated multi-microphone scenario of a single speaker in a reverberant environment with diffuse noise, which I simulate using gpurir [139]. For beamforming experiments, following the work of [140], I adopt the Voice Bank + DEMAND (VBD) dataset [141]. The training set is generated by mixing clean speech from 28 speakers of the Voice Bank corpus [142] with 10 types of noise from the DEMAND corpus [143] at four signal-to-noise ratio (SNR) levels (0, 5, 10, and 15 dB).

The test set uses the remaining 2 speakers and 5 noise types unseen during training, mixed at SNR values of 2.5, 7.5, 12.5, and 17.5 dB. All signals are sampled at 8 kHz following [135]. For each sample, I select a room size whose dimension are defined by uniforms distribution over width, height and depth $U(3,6) \times U(3,6) \times U(3,6)$ meters, a reverberation time $T_{60} \sim U(0.2,0.8)$ seconds, one noise recording, and the positions of six microphones and the source, without imposing constraints on their relative distances. Following the original implementation [135], the network is trained using the scale-invariant signal-to-distortion ratio between the undistorted signal and the one reconstructed by the beamforming pipeline. I use the Adam optimizer, a learning rate of 10^{-4}, and early stopping for regularization with a patience of 2 epochs. The capabilities of my pipeline are measured using the STOI metric as defined in [144].

For the SAD experiments, I start from the 121 noise types available by combining the NoiseX-92 [145] dataset and DEMAND. To further enrich the variability of background conditions, I augment the noise by introducing composite noise types mixing pairs of noises drawn from the two corpora, leading in principle to up to 121×60 possible combinations. Together with the Voice Bank speech corpus, this yields a dataset with the following statistics: training set of 15,034 samples (40.14% noise, 59.86% speech+noise), validation set of 1,726 samples (42.06% noise, 57.94% speech+noise), and test set of 1,300 samples (38.46% noise, 61.54% speech+noise). All signals are sampled at 8 kHz to be consistent with the beamforming pipeline. To train the SAD, I use the Adam optimizer with a learning rate of 10^{-4} and a binary cross-entropy loss. I evaluate the results on the test set, reporting accuracy, precision, and recall; the latter is particularly important, as the case of a false negative would mean that the present speech will be completely ignored by the whole pipeline.

All experiments use an STFT window length of $t_{stft} = 64$ ms and hop size of $t_{hop} = 15$ ms. Each input sequence spans 1 s of audio, corresponding to the total duration processed by the network (t_{mask}). For deployment on the GAP9 platform, buffer sizes are determined directly from these time windows and the signal's data rate.

Mixed-precision quantization

To reduce the memory requirements and computational cost of my deployed application, I convert the trained `float32` model into reduced-precision formats (`float16` or `int8`). I do so using GAPflow's Post-Training Quantization procedure, calibrating quantization ranges using a randomly selected subset of the training and validation data. However, the impact of quantization is not uniform across the different stages of my pipeline. In particular, the MVDR beamforming stage, which relies on covariance estimation and matrix inversion, is numerically ill-conditioned and thus especially sensitive to reduced precision [146].

As shown in Table 6.5, quantizing the MVDR stage to `int8` leads to complete degradation of the output, with STOI collapsing to zero. Using `float16` alleviates the problem but still results in a reduction of about 12% compared to the baseline, yielding a score even lower than the noisy input (0.861). By contrast, keeping MVDR and beamforming in `float32` fully preserves accuracy: in this case, the choice of precision for the G1_1 becomes largely irrelevant, with STOI values consistently around 0.878–0.884. Interestingly, the configuration with `int8` quantization for G1_1 slightly outperforms the float baselines, reaching 0.884, which represents an improvement of +0.023 over the noisy input.

These findings point toward a mixed-precision configuration, which I detail in the following subsection.

G1_2 + G2	G1_1 (Mask Estimator)		
	`int8`	`float16`	`float32`
`int8`	0.000	0.000	0.000
`float16`	0.746	0.766	0.759
`float32`	0.884	0.878	0.880

Table 6.5. STOI results for different quantization settings of G1_1 and G1_2 + G2. Baseline STOI for noisy input is 0.861.

Model	Mpar	Quantization	QAT	Device	Deployment	ms/inf	MAC/cyc	GMAC/J
TinyLSTM [67]	0.33	INT8	yes	STM32F746VE	N/A	4.26	0.36	0.14
TinyLSTM [67]	0.46	INT8	yes	STM32F746VE	N/A	2.39	0.36	0.14
RNNoise [69]	0.21	INT8	yes	STM32L476	NNoM w/ CMSIS-NN	3.28	0.45	1.84
LSTM256	1.24	MixFP16-INT8	no	8-core RISC-V	GAPFlow	2.50	2.11	17.78
GRU256	0.98	MixFP16-INT8	no	8-core RISC-V	GAPFlow	1.70	2.41	17.46
Ours	0.20	MixFP32-INT8	no	8-core RISC-V	GAPFlow	3.8	**3.10**	**20.20**

Table 6.6. Comparison with other MCU-deployed SE pipelines.

Deployment on GAP9 SoC

Following the results of the previous evaluation, I deployed the pipeline in a mixed configuration: the neural network is executed fully in `int8` quantization, while the remaining stages of the pipeline use the `float32` data type. As I can see from Figure 6.6, the G1 block of my pipeline is the most memory hungry, consuming about 3.14 MB of L3 memory, accounting also for the audio signals and their STFT, and about 920 kB of L2 memory during its execution. To this figure, I need to add the permanent memory needed by the application code, 279 kB, and the G2 inputs, stored in L2 for fast access, and the G1_1 weights, 20 kB, which amounts to 1248 kB. In this memory configuration and considering `int8` quantization, the mask estimation network requires 176M cycles (1.93 MACs/cycle), a 2.3× efficiency gain over `float16`, and, crucially, reduces the execution time below 1s of audio being processed. With the NE16 accelerator, efficiency further improves to 3.1 MACs/cycle, bringing total runtime down to 203 ms. Including the MVDR algorithm, which requires 13 million cycles at an efficiency of 0.49 MACs/cycle, and the STFT, which requires 2.4 million cycles at 3.1 MACs/cycle, the estimation of the beamforming weights takes about 244 ms. The last step of the pipeline, then, which is the effective beamforming step, requires an additional 1.4 million cycles at 0.9 MACs/cycle, resulting in a total of 248 ms for the complete neural beamforming.

Since recalculating weights at every t_{hop} would violate real-time constraints, I adopt an interleaved strategy in which G1 runs in the background while G2 continues with the most recently available weights. However, with the current memory configuration, considering that the G2 network requires 309 kB of L2 memory to be executed, this would be impossible, but if one accounts only for the peak L2 utilization of G1. In practice, though, the peak of 920 kB occurs only within individual layers of G1, while, between layers, the usage drops to 826 kB. This limits utilization to 1154 kB, allowing for the interleaved execution of G2.

The interleaved execution of the two elements of my pipeline limits the effective latency of the system to t_{hop} plus the execution time of G2, 19 ms, below the 20 ms requirement [133]. However, because G1 now progresses alongside G2, the time needed to produce a complete new set of weights increases from 244 ms to about 320 ms. Nevertheless, by reusing weights during this interval, the beamformer produces enhanced output continuously in real time.

To assess the robustness of the proposed interleaved strategy, I conducted an evaluation of how temporal delays affect the performance of my speech enhancement model. For this analysis, I selected 219 samples from the original test set of 800, as only these contained sufficiently long continuous speech segments suitable for the experiment. As illustrated in Figure 6.7, introducing a 300 ms delay between weight

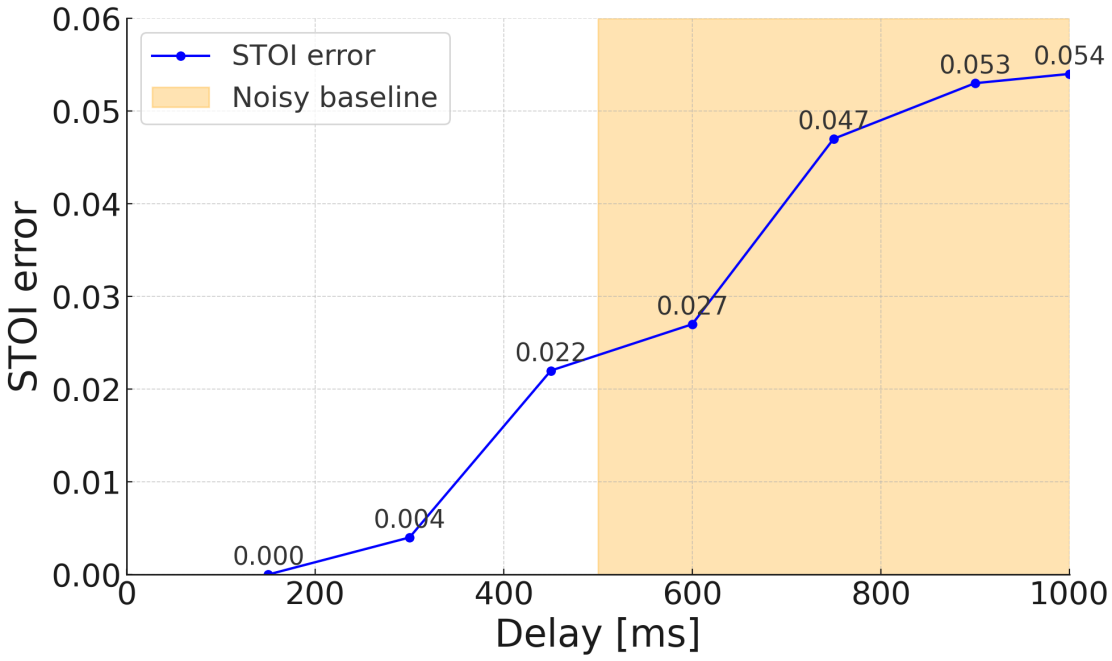

Figure 6.7. Effect of the delay on the STOI metric. The orange area indicates where the STOI of the models falls below that of the unprocessed noisy signal.

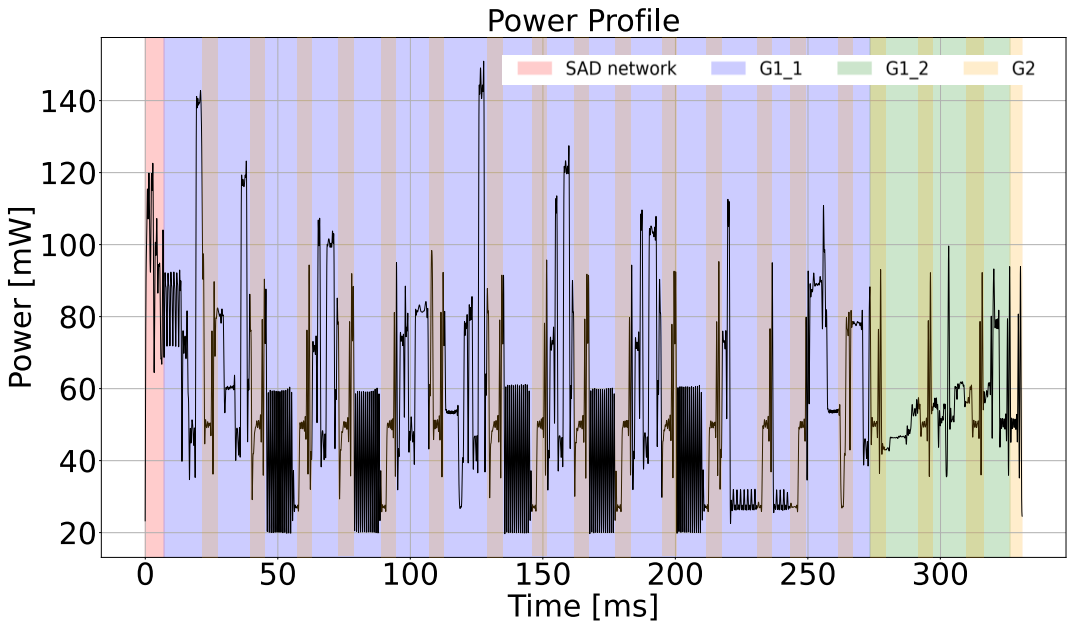

Figure 6.8. Power profile of the beamforming application.

estimation and beamforming results in only a 0.4% STOI degradation. This confirms, in line with prior findings[140], that modest delays between spatial parameter estimation and enhancement are tolerable. At the same time, the results highlight the importance of hardware acceleration: without it, the total execution time would reach approximately 600 ms, thereby reducing beamforming performance to a level below that of the unprocessed noisy input.

I deployed the whole pipeline end-to-end on a GAP9 EVK evaluation kit. In Figure 6.8, I report the measured power profile of the interleaved pipeline, acquired with Nordic's Power Profiler Kit.

System-level evaluation

I consider the deployment of my beamforming pipeline on a hearable device powered by a 100 mAh, 3.7 V Li-Po battery. This corresponds to an energy budget of approximately 1330 J and, at 17.28 mJ per inference, 13.2 mJ for the beamforming weights estimation, while the leftover 4.08 mJ is consumed by the application of the beamforming weights , providing about 6.85 hours of continuous execution. This falls short of the typical waking period of around 16 hours, reflecting a usage scenario in which the hearable is worn continuously throughout the day, as would be expected for users relying on it for hearing support.

However, typical users are not engaged in speech for more than, on average, 30% of their time [134]. For this reason, I integrate the SAD model (int8, 2.6M cycles at 29.1 MACs/cycle) to gate the pipeline execution. The int8 quantization does not reduce the accuracy of this network, which reaches 98.5% with a precision of 97.8% and a recall of 99.9%. Including the energy consumption of six 0.9 mW microphones and the inference cost of the SAD network, 0.62 mJ, the battery lifetime of a smart hearable device running my beamforming pipeline is extended up to 16 hours, reaching a target duration compatible with a hearable device operating for a full day.

Comparison with State-of-the-Art

Table 6.6 compares my solution with state-of-the-art speech enhancement approaches on MCUs. The baselines include *TinyLSTM* [67], evaluated on an STM32F7 MCU, and *RNNoise* [69], deployed on a low-power STM32L4 using the NNoM framework with a CMSIS-NN backend [147]. Both methods run on single-core devices with 8-bit quantization, which proves effective thanks to quantization-aware training (QAT) and model constraints that limit the numerical range of intermediate activations. I also compare against the work of Rusci et al. [70], which targets the same GAP9 platform used by us.

Unlike these baselines, my solution processes six input streams while retaining part of its computation in `float32`. Despite this more demanding setting, I achieve efficiency gains of up to 6.9×, 8.6×, 1.29× and 1.46× in MAC/cycle over *RNNoise*, *TinyLSTM*,*GRU256* and *LSTM256*, respectively. These improvements are enabled by my quantization strategy together with the exploitation of the hardware accelerator.

In terms of computational efficiency, my method delivers up to 10.98×, 144.3×, 1.16×, and 1.14×higher performance than *RNNoise*, *TinyLSTM*, *GRU256*, and *LSTM256*, respectively. Crucially, these gains are obtained *without* any degradation in accuracy compared to the full-precision model, and *without* relying on computationally expensive QAT procedures.

Conclusions

This thesis presented a hardware–software co-design methodology for deploying DNNs on heterogeneous MCU systems targeting extreme-edge applications. The proposed approach systematically combines algorithmic adaptation, workload partitioning, and hardware exploitation to achieve efficient inference under severe power, memory, and latency constraints.

The first part of this thesis analyzed the scaling behavior of neural networks on resource-constrained multicore MCUs, quantifying the trade-offs between model complexity, latency, and accuracy. Experiments conducted on the GAP8 platform demonstrated the limitations of homogeneous architectures when executing perception tasks of practical relevance. In particular, the MobileNetV2-SSD detector, running fully on-board a nano-drone prototype, sustained up to 1.6 frames/s at 134 mW, achieving 50 % mAP on two target classes and a 90 % mean detection rate across closed-loop trials. These results enabled the first fully autonomous nano-drone performing exploration, collision avoidance, and real-time object detection solely through on-board computation. However, further tests in cluttered environments revealed that combining visual collision avoidance and high-accuracy detection saturated the available compute throughput (1.6 Hz), falling short of the $\geq$8 Hz rate required for robust navigation. This finding emphasized the need for more capable, heterogeneous MCU architectures to sustain concurrent multi-network inference under tight power budgets.

Building on this motivation, the thesis investigated heterogeneous embedded systems through the GAP9 SoC, integrating RISC-V general-purpose cores with a convolutional accelerator. The case study on on-device pest detection highlighted the advantages of hardware specialization for energy-efficient AI at the edge. The Viola–Jones and MobileNetV3-SSD detectors achieved comparable detection rates (81.7 % and 83 %, respectively) on the first dataset, while the CNN-based model generalized significantly better (+27%) on a second dataset containing smaller insects. On GAP9, Viola–Jones completed inference in 221 ms for 320×240 images, 1.47× faster than on GAP8 due to higher clock speed and memory capacity. The CNN detector achieved 147 ms per inference, corresponding to a 9.5× speed-up over

GAP8 and an effective throughput of 10.9 MAC/cycle, demonstrating the substantial benefit of the integrated accelerator. System-level estimates further showed that a duty-cycled pest detection node could operate autonomously for approximately 199 days on a 1000 mAh battery, confirming the feasibility of long-term, low-power deployment.

Finally, the co-design framework was extended to application-specific workloads on heterogeneous MCUs, including VOD and speech enhancement (SE). The MR2-ByteTrack VOD pipeline improved detection accuracy by 2.16 % and reduced energy consumption by 43.6 %, while the SE pipeline achieved real-time neural beamforming with 20 ms latency at 17.28 mJ per inference, demonstrating that thanks to the provided metodology execution of complex neural networks can be sped up significantly and improving real times constraints on MCUs, while the different applications validate the generality of the methodology across sensing modalities.

Future Works

The results of this thesis have demonstrated the potential of hardware–software co-design to unlock efficient deep learning on heterogeneous MCU platforms. However, the observed limited performance and the increasing algorithmic complexity of modern models suggest that additional progress is both possible and necessary.

Future research should thus explore the next generation of neural architectures and hardware accelerators to further close the gap between embedded and cloud-level intelligence. The integration of more advanced hardware accelerators, from systolic arrays to in-memory computing units, could enable the execution of richer models under the same power envelope, extending the methodology developed in this work toward truly pervasive edge intelligence.

On the algorithmic side, incorporating sparsity-aware, modular, and dynamically adaptable network topologies will allow models to adjust their computational demand based on context, thereby maintaining energy proportionality without sacrificing inference quality. By leveraging these advances, future edge systems may progressively approximate the semantic richness and versatility characteristic of foundation models [148].

Expanding the range of application domains will also be key to validating this vision. Future embedded systems could leverage the proposed framework in human–machine interaction, enabling natural, low-latency multimodal interfaces that interpret speech, gestures, and visual context directly on-device, minimizing latency, preserving privacy, and reducing dependency on external connectivity. Ultimately, by combining architectural specialization with adaptive intelligence, future embedded systems may deliver the semantic richness of modern AI “at the fingertip of the user.”

Bibliography

[1] A. Guarnieri, S. Maini, G. Molari, V. Rondelli *et al.*, "Automatic trap for moth detection in integrated pest management," *Bulletin of Insectology*, vol. 64, no. 2, pp. 247–251, 2011.

[2] D. Brunelli, T. Polonelli, and L. Benini, "Ultra-low energy pest detection for smart agriculture," in *2020 IEEE SENSORS*, 2020, pp. 1–4.

[3] RangiLyu, "Nanodet-plus superfast and high accuracy lightweight anchor-free object detection model," 2021. [Online]. Available: https://github.com/RangiLyu/nanodet

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," vol. 521, no. 7553, pp. 436–444. [Online]. Available: https://doi.org/10.1038/nature14539

[5] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157 vol.2.

[6] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 886–893 vol. 1.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[9] W. Xiong, J. Droppo, X. Huang, F. Seide, M. L. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Toward human parity in conversational speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 12, 2017.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Proceedings of Workshop at ICLR*, vol. 2013, 01 2013.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[12] V. Samborska, "Scaling up: how increasing inputs has made artificial intelligence more capable," *Our World in Data*, 2025, https://ourworldindata.org/scaling-up-ai.

[13] N. P. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. Patterson, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023. [Online]. Available: https://arxiv.org/abs/2304.01433

[14] L. Lamberti, "Enabling multi-tasking ai-based perception on autonomous nano-uavs," Ph.D. dissertation, alma, Luglio 2024. [Online]. Available: https://amsdottorato.unibo.it/id/eprint/11643/

[15] Z. Rasheed, S. Ghwanmeh, and A. Almahadin, "A critical review of ai-driven ar glasses for realtime multilingual communication," in *2024 Advances in Science and Engineering Technology International Conferences (ASET)*, 2024, pp. 1–6.

[16] P. Busia, A. Cossettini, T. M. Ingolfsson, S. Benatti, A. Burrello, V. J. B. Jung, M. Scherer, M. A. Scrugli, A. Bernini, P. Ducouret, P. Ryvlin, P. Meloni, and L. Benini, "Reducing false alarms in wearable seizure detection with eegformer: A compact transformer model for mcus," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 18, no. 3, pp. 608–621, 2024.

[17] L. S. Karumbunathan, "Nvidia jetson agx orin series technical brief," NVIDIA Corporation, Technical Brief v1.2 TB_10749-001_v1.2, 2022, accessed: 10 October 2025. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf

[18] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64mw dnn-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, vol. PP, pp. 1–1, 05 2019.

[19] 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 *2021 IEEE International Symposium on Circuits and Systems*

(ISCAS), 2021, pp. 1–5.

[20] J. Suto, "Embedded system-based sticky paper trap with deep learning-based insect-counting algorithm," *Electronics*, vol. 10, no. 15, p. 1754, 2021.

[21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015.

[22] 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 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[23] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020.

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[25] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2018, pp. 4510–4520.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer International Publishing, 2014, pp. 740–755.

[27] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2020, pp. 10 778–10 787.

[28] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXivpreprintarXiv:2107.08430*, 2021.

[29] 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*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 213–229.

[30] S. Mehta and M. Rastegari, "MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=vh-0sUt8HlG

[31] ——, "Separable Self-attention for Mobile Vision Transformers," 2022. [Online]. Available: https://arxiv.org/abs/2206.02680

[32] S. N. Wadekar and A. Chaurasia, "MobileViTv3: Mobile-Friendly Vision

Transformer with Simple and Effective Fusion of Local, Global and Input Features," 2022. [Online]. Available: https://arxiv.org/abs/2209.15159

[33] L. Lamberti, V. Niculescu, M. Barcis, L. Bellone, E. Natalizio, L. Benini, and D. Palossi, "Tiny-pulp-dronets: Squeezing neural networks for faster and lighter inference on multi-tasking autonomous nano-drones," *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 287–290, 2022.

[34] R. J. Bouwmeester, F. Paredes-Vallés, and G. C. H. E. de Croon, "Nanoflownet: Real-time dense optical flow on a nano quadcopter," *arXiv preprint arXiv:2209.06918*, 2022.

[35] 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 ultra-low power autonomous flying nano-uavs," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[36] D. Palossi, F. Tombari, S. Salti, M. Ruggiero, L. Stefano, and L. Benini, "Gpu-shot: Parallel optimization for real-time 3d local description," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2013, pp. 584–591.

[37] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Lujan, S. Furber, A. J. Davison, P. H. J. Kelly, and M. F. P. O'Boyle, "Slambench2: Multi-objective head-to-head benchmarking for visual slam," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3637–3644.

[38] K. McGuire, G. de Croon, and K. Tuyls, "A comparative study of bug algorithms for robot navigation," *Robotics and Autonomous Systems*, vol. 121, p. 103261, 2019.

[39] Q.-l. Xu, "Randombug: Novel path planning algorithm in unknown environment," *The Open Electrical & Electronic Engineering Journal*, vol. 8, no. 1, 2014.

[40] E. Magid and E. Rivlin, "Cautiousbug: a competitive algorithm for sensory-based robot navigation," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2757–2762 vol.3, 2004.

[41] Q.-L. Xu and G.-Y. Tang, "Vectorization path planning for autonomous mobile agent in unknown environment," *Neural Computing and Applications*, vol. 23, no. 7, pp. 2129–2135, 2013.

[42] I. Kamon, E. Rivlin, and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, 1996, pp. 429–435.

[43] T. Q. Khoi, N. A. Quang, and N. K. Hieu, "Object detection for drones on raspberry pi potentials and challenges," *IOP Conference Series: Materials*

Science and Engineering, vol. 1109, no. 1, p. 012033, mar 2021.

[44] M. Preti, C. Moretti, G. Scarton, G. Giannotta, and S. Angeli, "Developing a smart trap prototype equipped with camera for tortricid pests remote monitoring," *Bulletin of insectology*, vol. 74, no. 1, pp. 147–160, 2021.

[45] A. Suárez, R. S. Molina, G. Ramponi, R. Petrino, L. Bollati, and D. Sequeiros, "Pest detection and classification to reduce pesticide use in fruit crops based on deep neural networks and image processing," in *2021 XIX Workshop on Information Processing and Control (RPIC)*. IEEE, 2021, pp. 1–6.

[46] J. Suto, "A novel plug-in board for remote insect monitoring," *Agriculture*, vol. 12, no. 11, p. 1897, 2022.

[47] A. Albanese, M. Nardello, and D. Brunelli, "Automated pest detection with dnn on the edge for precision agriculture," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 3, pp. 458–467, 2021.

[48] D. Čirjak, I. Aleksi, D. Lemic, and I. Pajač Živković, "Efficientdet-4 deep neural network-based remote monitoring of codling moth population for early damage detection in apple orchard," *Agriculture*, vol. 13, no. 5, p. 961, 2023.

[49] M. J. Schrader, P. Smytheman, E. H. Beers, and L. R. Khot, "An open-source low-cost imaging system plug-in for pheromone traps aiding remote insect pest population monitoring in fruit crops," *Machines*, vol. 10, no. 1, p. 52, 2022.

[50] M. Rusci, L. Bompani, O. Baldoni, C. Montanari, D. Brunelli, and L. Benini, "Parallel execution of the viola-jones algorithm on mcus for low-cost automated pest detection," in *2023 IEEE Conference on AgriFood Electronics (CAFE)*, 2023, pp. 35–39.

[51] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *2015 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2015, pp. 4489–4497.

[52] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei, "Flow-guided feature aggregation for video object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 408–417.

[53] Y. Lyu, M. Y. Yang, G. Vosselman, and G.-S. Xia, "Video object detection with a convolutional regression tracker," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 176, pp. 139–150, 2021.

[54] Z. Zhang, D. Cheng, X. Z. S. Lin, and J. Dai, "Integrated object detection and tracking with tracklet-conditioned detection," *ArXiv*, vol. abs/1811.11167, 2018.

[55] Y. Chen, Y. Cao, H. Hu, and L. Wang, "Memory enhanced global-local aggregation for video object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 06 2020, pp. 10 334–10 343.

[56] W. Han, P. Khorrami, T. L. Paine, P. Ramachandran, M. Babaeizadeh, H. Shi, J. Li, S. Yan, and T. S. Huang, "Seq-nms for video object

detection." *CoRR*, vol. abs/1602.08465, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1602.html#HanKPRBSLYH16

[57] H. Belhassen, H. Zhang, V. Fresse, and E.-B. Bourennane, "Improving video object detection by seq-bboxmatching." in *VISIGRAPP(5:VISAPP)*, 2019, pp. 226–233.

[58] M. Li, L. Li, R. Bai, J. Ren, B. Meng, and Y. Yang, "A motion-based seq-bbox matching method for video object detection," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, 2021, pp. 1–7.

[59] L. He, Q. Zhou, X. Li, L. Niu, G. Cheng, X. Li, W. Liu, Y. Tong, L. Ma, and L. Zhang, "End-to-end video object detection with spatial-temporal transformers," in *Proceedings of the 29th ACM International Conference on Multimedia*, ser. MM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1507–1516. [Online]. Available: https://doi.org/10.1145/3474085.3475285

[60] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable {detr}: Deformable transformers for end-to-end object detection," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=gZ9hCDWe6ke

[61] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2021, pp. 9992–10 002. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.00986

[62] Y. Shi, N. Wang, and X. Guo, "Yolov: Making still image object detectors great at video object detection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 2, pp. 2254–2262, Jun. 2023.

[63] M. Zhu, K. Han, E. Wu, Q. Zhang, Y. Nie, Z. Lan, and Y. Wang, "Dynamic resolution network," in *Neural Information Processing Systems*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235358802

[64] T. Verelst and T. Tuytelaars, "Blockcopy: High-resolution video processing with block-sparse feature propagation and online policies," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 5138–5147.

[65] M. Liu, M. Zhu, M. White, Y. Li, and D. Kalenichenko, "Looking fast and slow: Memory-guided mobile video object detection," *arXiv preprint arXiv:1903.10172*, 2019.

[66] B. A. Motetti, L. Crupi, M. O. M. E. Elshaigi, M. Risso, D. J. Pagliari, D. Palossi, and A. Burrello, "Adaptive deep learning for efficient visual pose estimation aboard ultra-low-power nano-drones," *ArXiv*, vol. abs/2401.15236, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267312457

[67] I. Fedorov, M. Stamenovic, C. R. Jensen, L.-C. Yang, A. Mandell, Y. Gan, M. Mattina, and P. N. Whatmough, "TinyLSTMs: Efficient neural speech enhancement for hearing aids," in *Interspeech*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218862718

[68] J. Ma, "A higher-level neural network library on microcontrollers (NNoM)," October 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4158710

[69] J.-M. Valin, "A hybrid DSP/deep learning approach to real-time full-band speech enhancement," *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–5, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:21722177

[70] M. Rusci, M. Fariselli, M. Croome, F. Paci, and E. Flamand, "Accelerating RNN-based speech enhancement on a multi-core MCU with mixed FP16-INT8 post-training quantization," in *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Cham: Springer Nature Switzerland, 2023, pp. 606–617.

[71] H. Yan, J. Zhang, C. Jiang, and S. Zhang, "LABNet: A lightweight attentive beamforming network for ad-hoc multichannel microphone invariant real-time speech enhancement," 2025. [Online]. Available: https://arxiv.org/abs/2507.16190

[72] A. Kovalyov, K. Patel, and I. Panahi, "DFSNet: A steerable neural beamformer invariant to microphone array configuration for real-time, low-latency speech enhancement," in *Proceedings of Interspeech 2023*, 2023, pp. 2493–2497.

[73] D. Lee and J.-W. Choi, "DeFTAN-II: Efficient multichannel speech enhancement with subgroup processing," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2024.

[74] X. Ren, X. Zhang, L. Chen, X. Zheng, C. Zhang, L. Guo, and B. Yu, "A causal U-Net based neural beamforming network for real-time multi-channel speech enhancement," 08 2021, pp. 1832–1836.

[75] J.-H. Wang, P. T. Le, S.-J. Kuo, T.-C. Tai, K.-C. Li, S.-L. Chen, Z.-Y. Wang, T. Pham, Y.-H. Li, and J.-C. Wang, "Audio pre-processing and beamforming implementation on embedded systems," *Electronics*, vol. 13, no. 14, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/14/2784

[76] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[77] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[78] J. Bai, F. Lu, K. Zhang *et al.*, "Onnx: Open neural network exchange," https://github.com/onnx/onnx, 2019.

[79] 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.*, 2021.

[80] M. Scherer, L. Macan, V. J. B. Jung, P. Wiese, L. Bompani, A. Burrello, F. Conti, and L. Benini, "Deeploy: Enabling energy-efficient deployment of small language models on heterogeneous microcontrollers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 4009–4020, 2024.

[81] J. Van Delm, M. Vandersteegen, A. Burrello, G. M. Sarda, F. Conti, D. J. Pagliari *et al.*, "HTVM: Efficient Neural Network Deployment On Heterogeneous TinyML Platforms," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[82] M. A. Hamdi, F. Daghero, G. M. Sarda, J. V. Delm, A. Symons, L. Benini *et al.*, "Match: Model-aware tvm-based compilation for heterogeneous edge devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2025.

[83] Z. Chen, C. H. Yu, T. Morris, J. Tuyls, Y.-H. Lai, J. Roesch *et al.*, "Bring Your Own Codegen to Deep Learning Compiler," *arXiv:2105.03215*, 2021.

[84] M. Maas, U. Beaugnon, A. Chauhan, and B. Ilbeyi, "Telamalloc: Efficient on-chip memory allocation for production machine learning accelerators," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS 2023, 2022, p. 123–137.

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

[86] D. Palossi, A. Gomez, S. Draskovic, K. Keller, L. Benini, and L. Thiele, "Self-sustainability in nano unmanned aerial vehicles: A blimp case study," in *Proceedings of the computing frontiers conference*, 2017, pp. 79–88.

[87] A. Birk, B. Wiggerich, H. Bülow, M. Pfingsthorn, and S. Schwertfeger, "Safety, Security, and Rescue Missions with an Unmanned Aerial Vehicle (UAV)," *Journal of Intelligent & Robotic Systems*, vol. 64, no. 1, pp. 57–76, October 2011.

[88] M. Giurfa and R. Menzel, "Insect visual perception: complex abilities of simple nervous systems," *Current Opinion in Neurobiology*, vol. 7, no. 4, pp. 505–513, 1997.

[89] B. Webb, "The internal maps of insects," *Journal of Experimental Biology*, vol. 222, 2019.

[90] R. J. Wood, B. Finio, M. Karpelson, K. Ma, N. O. Pérez-Arancibia, P. S. Sreetharan, H. Tanaka, and J. P. Whitney, "Progress on "Pico" air vehicles," in *Robotics Research : The 15th International Symposium ISRR*, H. I. Christensen and O. Khatib, Eds. Cham: Springer International Publishing, 2017, pp. 3–19.

[91] I. Dinstein, U. Hasson, N. Rubin, and D. J. Heeger, "Brain areas selective for both observed and executed movements," *Journal of neurophysiology*, vol. 98, no. 3, pp. 1415–1427, 2007.

[92] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[93] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov *et al.*, "The open images dataset v4," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[94] B. P. Duisterhof, S. Li, J. Burgu'es, V. J. Reddi, and G. C. de Croon, "Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9099–9106, 2021.

[95] D. Ju, D. Mota-Sanchez, E. Fuentes-Contreras, Y.-L. Zhang, X.-Q. Wang, and X.-Q. Yang, "Insecticide resistance in the cydia pomonella (l): Global status, mechanisms, and research directions," *Pesticide Biochemistry and Physiology*, vol. 178, p. 104925, 2021.

[96] F. Wan, C. Yin, R. Tang, M. Chen, Q. Wu, C. Huang, W. Qian, O. Rota-Stabelli, N. Yang, S. Wang *et al.*, "A chromosome-level genome assembly of cydia pomonella provides insights into chemical ecology and insecticide resistance," *Nature Communications*, vol. 10, no. 1, p. 4237, 2019.

[97] E. Roma, V. A. Laudicina, M. Vallone, and P. Catania, "Application of precision agriculture for the sustainable management of fertilization in olive groves," *Agronomy*, vol. 13, no. 2, 2023.

[98] G. Kamyshova, A. Osipov, S. Gataullin, S. Korchagin, S. Ignar, T. Gataullin, N. Terekhova, and S. Suvorov, "Artificial neural networks and computer vision's-based phytoindication systems for variable rate irrigation improving," *IEEE Access*, vol. PP, pp. 1–1, 01 2022.

[99] J. Suto, "Codling moth monitoring with camera-equipped automated traps: A review," *Agriculture*, vol. 12, no. 10, p. 1721, 2022.

[100] A. C. Teixeira, J. Ribeiro, R. Morais, J. J. Sousa, and A. Cunha, "A systematic review on automatic insect detection using deep learning," *Agriculture*, vol. 13, no. 3, 2023.

[101] O. López Granado, M. Martínez-Rach, H. Migallón, M. Malumbres, A. Bonastre Pina, and J. Serrano Martín, "Monitoring pest insect traps by means of low-power image sensor technologies," *Sensors (Basel, Switzerland)*, vol. 12, pp. 15 801–19, 12 2012.

[102] M. Saari, A. M. bin Baharudin, P. Sillberg, S. Hyrynsalmi, and W. Yan, "Lora — a survey of recent research trends," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 0872–0877.

[103] D. Singh, S. Sai Prashanth, S. Kundu, and A. Pal, "Low-power microcontroller for wireless sensor networks," in *TENCON 2009 - 2009 IEEE Region 10 Conference*, 2009, pp. 1–6.

[104] 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 Journal of Solid-State Circuits*, vol. 59, no. 1, pp. 128–142, 2024.

[105] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1. Ieee, 2001, pp. I–I.

[106] B. Luca, F. Alberto, J. Mauro, P. Mauro, and A. Cesare, "Precise agriculture: Effective deep learning strategies to detect pest insects," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. JAS-2021-0798, p. 246, 2022.

[107] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020.

[108] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," 2019.

[109] M. Baldessari, C. Rizzi, G. Tolotti, and G. Angeli, "Evaluation of an aerosol emitter for mating disruption of cydia pomonella in italy," *Communications in agricultural and applied biological sciences*, vol. 78, pp. 267–71, 01 2013.

[110] B. Giovanni, L. Andrea, D. Thomson, and C. Ioriatti, "Sex pheromone aerosol devices for mating disruption: Challenges for a brighter future," *Insects*, vol. 10, p. 308, 09 2019.

[111] P. L. Shaffer and H. J. Gold, "A simulation model of population dynamics of the codling moth, cydia pomonella," *Ecological Modelling*, vol. 30, pp. 247–274, 1985.

[112] P. Damos and N. Kouloussis, "A degree-day phenological model for cydia pomonella and its validation in a mediterranean climate," *Bulletin of Insectology*, vol. 71, 04 2018.

[113] V. P. Jones, R. Hilton, J. F. Brunner, W. J. Bentley, D. G. Alston, B. Barrett, R. A. Van Steenwyk, L. A. Hull, J. F. Walgenbach, W. W. Coates, and T. J. Smith, "Predicting the emergence of the codling moth, cydia pomonella (lepidoptera: Tortricidae), on a degree-day scale in north america," *Pest Management Science*, vol. 69, no. 12, pp. 1393–1398, 2013.

[114] V. M. Suresh, R. Sidhu, P. Karkare, A. Patil, Z. Lei, and A. Basu, "Powering the iot through embedded machine learning and lora," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, 2018, pp. 349–354.

[115] S. Gutiérrez, I. Martínez, J. Varona, M. Cardona, and R. Espinosa, "Smart mobile lora agriculture system based on internet of things," in *2019 IEEE 39th Central America and Panama Convention (CONCAPAN XXXIX)*, 2019, pp. 1–6.

[116] B. P. Rimal, D. P. Van, and M. Maier, "Mobile edge computing empowered fiber-wireless access networks in the 5g era," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 192–200, 2017.

[117] C. Hu, W. Shi, C. Li, J. Sun, D. Wang, J. Wu, and G. Tang, "Impact of light and shadow on robustness of deep neural networks," 2023.

[118] I. Pajač Živković and B. Barić, "The behaviour of codling moth (lepidoptera: Tortricidae) in the croatian apple orchards," *IOBC-WPRS Bulletin*, vol. 74, pp. 79–82, 01 2012.

[119] E. Sola-Thomas and M. H. Imtiaz, "An ultra-low-power design of smart wearable stereo camera," in *SoutheastCon 2021*, 2021, pp. 1–8.

[120] L. Lamberti, L. Bompani, V. J. Kartsch, M. Rusci, D. Palossi, and L. Benini, "Bio-inspired autonomous exploration policies with cnn-based object detection on nano-drones," in *2023 Design, Automation & Testin Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[121] E. AlNuaimi, E. Cereda, R. Psiakis, S. Sugumar, A. Giusti, and D. Palossi, "A deep learning-based face mask detector for autonomous nano-drones (student abstract)," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, 2022, pp. 12 903–12 904.

[122] J. Portilla, G. Mujica, J.-S. Lee, and T. Riesgo, "The extreme edge at the bottom of the internet of things: A review," *IEEE Sensors Journal*, vol. PP, pp. 1–1, 01 2019.

[123] 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 Journal of Solid-State Circuits*, vol. 57, no. 1, pp. 127–139, 2022.

[124] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018, cite arxiv:1804.02767Comment: Tech Report. [Online]. Available: http://arxiv.org/abs/1804.02767

[125] Y. Zhang, P. Sun, Y. Jiang, D. Yu, Z. Yuan, P. Luo, W. Liu, and X. Wang,

"Bytetrack: Multi-object tracking by associating every detection box," in *European Conference on Computer Vision*, 2021.

[126] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[127] F.-J. Chang, M. Radfar, A. Mouchtaris, and M. Omologo, "Multi-channel transformer transducer for speech recognition," 08 2021, pp. 296–300.

[128] M. Ohlenbusch, C. Rollwage, and S. Doclo, "Multi-microphone noise data augmentation for DNN-based own voice reconstruction for hearables in noisy environments," 04 2024, pp. 416–420.

[129] N. L. Westhausen, H. Kayser, T. Jansen, and B. T. Meyer, "Real-time multichannel deep speech enhancement in hearing aids: Comparing monaural and binaural processing in complex acoustic scenarios," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 32, p. 4596–4606, October 2024. [Online]. Available: https://doi.org/10.1109/TASLP.2024.3473315

[130] P. U. Diehl, H. Zilly, F. Sattler, Y. Singer, K. Kepp, M. Berry, H. Hasemann, M. Zippel, M. Kaya, P. Meyer-Rachner, A. Pudszuhn, V. M. Hofmann, M. Vormann, and E. Sprengel, "Deep learning-based denoising streamed from mobile phones improves speech-in-noise understanding for hearing aid users," *Frontiers in Medical Engineering*, vol. Volume 1 - 2023, 2023. [Online]. Available: https://www.frontiersin.org/journals/medical-engineering/articles/10.3389/fmede.2023.1281904

[131] A. Li, W. Liu, C. Zheng, and X. Li, "Embedding and beamforming: All-neural causal beamformer for multichannel speech enhancement," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 6487–6491.

[132] J. Pan, P. Shen, H. Zhang, and X. Zhang, "Efficient multi-channel speech enhancement with spherical harmonics injection for directional encoding," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 9561–9565.

[133] M. Stone and B. Moore, "Tolerable hearing aid delays. ii. estimation of limits imposed during speech production," *Ear and hearing*, vol. 23, pp. 325–38, 09 2002.

[134] K. Smeds, S. Gotowiec, F. Wolters, P. Herrlin, J. Larsson, and M. Dahlquist, "Selecting scenarios for hearing-related laboratory testing," *Ear and Hearing*, vol. 41, 2020. [Online]. Available: https://journals.lww.com/ear-hearing/fulltext/2020/11001/selecting_scenarios_for_hearing_related_laboratory.3.aspx

[135] E. Ceolini and S.-C. Liu, "Combining deep neural networks and beamforming for real-time multi-channel speech enhancement using a wireless acoustic sensor network," in *2019 IEEE 29th International Workshop on Machine*

Learning for Signal Processing (MLSP), 2019, pp. 1–6.

[136] S. Markovich-Golan, W. Kellermann, and S. Gannot, *Spatial Filtering*. John Wiley & Sons, Ltd, 2018, ch. 10, pp. 189–217. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119279860.ch10

[137] H. Erdogan, J. Hershey, S. Watanabe, M. Mandel, and J. Le Roux, "Improved MVDR beamforming using single-channel mask prediction networks," 09 2016, pp. 1981–1985.

[138] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '16. IEEE, June 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459

[139] D. Diaz-Guerra, A. Miguel, and J. R. Beltran, "gpuRIR: A python library for room impulse response simulation with GPU acceleration," *Multimedia Tools Appl.*, vol. 80, no. 4, p. 5653–5671, February 2021. [Online]. Available: https://doi.org/10.1007/s11042-020-09905-3

[140] L. Cheng, A. Pandey, B. Xu, T. Delbruck, V. Ithapu, and S.-C. Liu, "Modulating state space model with slowfast framework for compute-efficient ultra low-latency speech enhancement," 04 2025, pp. 1–5.

[141] H. Yen, F. Germain, G. Wichern, and J. Le Roux, "Cold diffusion for speech enhancement," 06 2023, pp. 1–5.

[142] C. Veaux, J. Yamagishi, and S. King, "The voice bank corpus: Design, collection and data analysis of a large regional accent speech database," in *2013 International Conference Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)*, 2013, pp. 1–4.

[143] J. Thiemann, N. Ito, and E. Vincent, "The diverse environments multi-channel acoustic noise database (DEMAND): A database of multichannel environmental noise recordings," *The Journal of the Acoustical Society of America*, vol. 133, p. 3591, 05 2013.

[144] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, "A short-time objective intelligibility measure for time-frequency weighted noisy speech," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2010, pp. 4214–4217.

[145] A. Varga and H. J. Steeneken, "Assessment for automatic speech recognition: II. NOISEX-92: A database and an experiment to study the effect of additive noise on speech recognition systems," *Speech Communication*, vol. 12, no. 3, pp. 247–251, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167639393900953

[146] S. Zhao and D. Jones, "A fast-converging adaptive frequency-domain MVDR beamformer for speech enhancement," vol. 3, 09 2012.

[147] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm Cortex-M CPUs," *ArXiv*, vol. abs/1801.06601, 2018. [Online].

Available: https://api.semanticscholar.org/CorpusID:691340

[148] R. Bommasani, D. Hudson, E. Adeli, R. Altman, S. Arora, S. Arx, M. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Davis, D. Demszky, and P. Liang, "On the opportunities and risks of foundation models," 08 2021.