A Machine Learning-Oriented Survey on Tiny Machine Learning

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 January 2024, accepted 4 February 2024, date of publication 12 February 2024, date of current version 16 February 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3365349*

SURVEY

# A Machine Learning-Oriented Survey on Tiny Machine Learning

**LUIGI CAPOGROSSO[1], (Student Member, IEEE), FEDERICO CUNICO[1], (Graduate Student Member, IEEE), DONG SEON CHENG[1], FRANCO FUMMI[1], (Member, IEEE), AND MARCO CRISTANI[1,2], (Member, IEEE)**

[1]Department of Engineering for Innovation Medicine, University of Verona, 37134 Verona, Italy
[2]QUALYCO S.r.l, Spin-off of the University of Verona, 37134 Verona, Italy

Corresponding author: Luigi Capogrosso (luigi.capogrosso@univr.it)

This work was supported by the Piano Nazionale di Ripresa e Resilienza (PNRR) research activities of the consortium Interconnected North-Est Innovation Ecosystem (iNEST) funded by the European Union Next-GenerationEU PNRR–Missione 4 Componente 2, Investimento 1.5–D.D. 1058, 23 July 2022, under Grant ECS_00000043. This manuscript reflects only the Authors' views and opinions. Neither the European Union nor the European Commission can be considered responsible for them.

**ABSTRACT** The emergence of Tiny Machine Learning (TinyML) has positively revolutionized the field of Artificial Intelligence by promoting the joint design of resource-constrained IoT hardware devices and their learning-based software architectures. TinyML carries an essential role within the fourth and fifth industrial revolutions in helping societies, economies, and individuals employ effective AI-infused computing technologies (e.g., smart cities, automotive, and medical robotics). Given its multidisciplinary nature, the field of TinyML has been approached from many different angles: this comprehensive survey wishes to provide an up-to-date overview focused on all the learning algorithms within TinyML-based solutions. The survey is based on the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) methodological flow, allowing for a systematic and complete literature survey. In particular, firstly, we will examine the three different workflows for implementing a TinyML-based system, i.e., ML-oriented, HW-oriented, and co-design. Secondly, we propose a taxonomy that covers the learning panorama under the TinyML lens, examining in detail the different families of model optimization and design, as well as the state-of-the-art learning techniques. Thirdly, this survey will present the distinct features of hardware devices and software tools that represent the current state-of-the-art for TinyML intelligent edge applications. Finally, we discuss the challenges and future directions.

**INDEX TERMS** TinyML, edge intelligence, efficient deep learning, embedded systems.

## I. INTRODUCTION

Over the past decades, a prodigious amount of research has been invested in improving embedded technologies to enable real-time solutions for many complex and safety-critical applications [1]. In this regard, hardware-specific (e.g., Edge TPUs) and Micro-Controller Unit (MCU)-based embedded systems have earned a lot of attention, primarily due to their low power requirements and high performance, and secondarily for their maintainability, adaptability, and reliability [2]. Their integration with sensors enables the perception of the external world, their connection with activators allows different kinds of interventions, and their interconnection unlocks distributed intelligence.

Embedded technologies are essentially the pillars of the Internet of Things (IoT) [3] and the associated *smart-X* applications: *smart buildings* [4] and *cities* [5], *smart metering* [6], *agriculture* [7] and *environment* [8], *smart health* [9], *smart logistics* [10], and *smart retail* [11]. More recent advances in the Industrial Internet of Things (IIoT) [12] have facilitated the real-time intelligent processing of massive amounts of data, promoting fields such as *autonomous driving* [13], *smart factories* [14], *anomaly detection* [15], and *predictive maintenance* [16].

The associate editor coordinating the review of this manuscript and approving it for publication was Ze Ji.

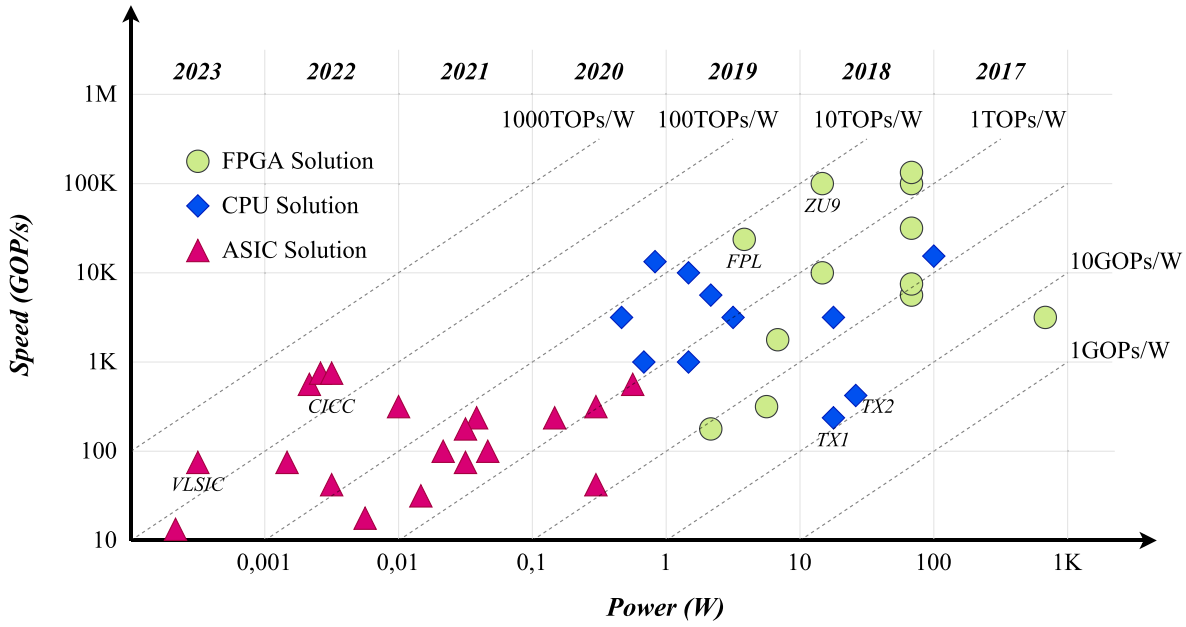



**FIGURE 1.** A glance at the latest hardware developed for TinyML reveals a notable trend: recent advances focus on minimizing power consumption. This implies that the primary emphasis is enabling ML to run on devices with limited resources.

When we talk about the intelligence of onboard embedded technologies, we mean the learning algorithms that allow devices to make reasoned decisions based on acquired data. Unfortunately, Machine Learning (ML) on tiny devices is substantially hard due to severe architectural, energetic, and latency constraints [17]: the available memory averages a few kilobytes, the accessible power is in the order of milliwatts, and often real-time responses must be guaranteed [18], as in safety-critical systems like health care devices, autonomous driving, or human-robot collaboration in industrial environments, where delayed decisions may have disastrous consequences, ranging from compromised patient well-being and increased road safety hazards, to operational disruptions.

From these premises, since 2018, the notion of ***Tiny Machine Learning (TinyML)*** has begun to take shape with the following acknowledged definition: *TinyML is a paradigm that facilitates running ML on the edge devices with minimal processor and memory requirements; hence, the power consumption of such systems is expected to be within a few milliwatts or less* [19]. Based on our literature search, the hardware requirements for memory footprint are below 1 MB, mostly between 64 KB and 256 KB of memory. In terms of performance, the computing units on board of such devices are usually in the range of 40 to 400 MHz.

The challenges for TinyML practitioners are formidable: e.g., in modern neural networks, among the best currently available technologies, the number of required parameters have skyrocketed to the order of billions [20], with larger networks having better results and broader applicability. Unfortunately, the energy required to run these networks is proportional to their size, making this trend of scaling up neural networks energetically unsustainable at large scales [21]: another reason why TinyML has to be considered as a necessary, other than promising, research direction. In this sense, TinyML allows developers to deploy complex algorithms in really constrained devices that are limited but also cheap, frugal, and portable, making them attractive for many use cases. Recent market trends (see Figure 1) confirm this rationale: priority has been given to deploying hardware that is less power-hungry and constraining: truly, TinyML has to be tinier.

When it comes to developing a TinyML solution, there are two main classical workflows, namely ***ML-oriented*** and ***HW-oriented***, and a third more recent approach, ***co-design***. The classical workflows are widely adopted and separate the ML framework design from its hardware incarnation [1], [22]. In the first approach, ML experts create, train, and test a suitable model for the problem domain, optimize its parameters, and then deploy this solution on a satisfactory device. In the second one, the hardware platform is not prearranged, and development aims to produce optimized hardware by employing specially reduced models and techniques.

The novel workflow is named co-design because ML experts and hardware engineers are involved together from the start in the solution design and actively exchange operational knowledge [23]. Hardware engineers approach the mathematical notions underlying the ML algorithms and propose suitable hardware components for efficient translations. Meanwhile, ML researchers examine the cutting-edge resources they can exploit and potentially re-design their algorithms to seamlessly integrate hardware and software, where the form and content are malleable and shape each other. Here is where shape and content are mixed together,

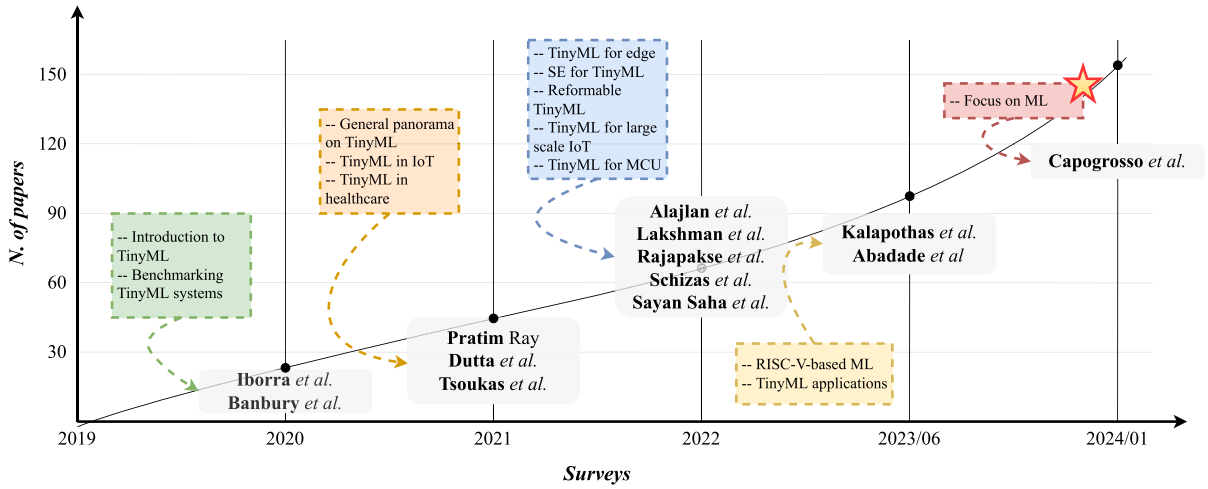



**FIGURE 2.** The number of papers on TinyML published so far and the surveys on the topic. As evident, there is an exponential growth in the number of research papers, and it's worth noting that our survey not only stands as the most recent but also uniquely concentrates on the ML perspective, distinguishing it from all other existing surveys in the field.

with a blended recipe that constitutes the state-of-the-art of contemporary TinyML. Specifically, in Section IV, we will detail each of these workflows.

### A. MOTIVATION AND CONTRIBUTIONS

We offer two contributions with this survey: we first provide an up-to-date overview of the rapidly evolving state-of-the-art in the field of TinyML. Since the number of research articles published on TinyML is increasing exponentially (see Figure 2), the number of surveys and papers on the subject is following suit. Specifically, we catalog the literature up to January 2024.

As a further and unique contribution, this survey emphasizes the ML point of view, not only reporting the very latest in TinyML frameworks but also suggesting recent variations and advancements in the ML technologies that a TinyML practitioner may want to explore to improve on the state-of-the-art: e.g., topics like meta-learning [24], Rational Activation Functions (RAFs) [25], and Versatile Learned Optimizers (VeLO) [26]. In this regard, we provide insights into these ML methodologies and address the most recent developments as potential TinyML future breakthroughs.

Readers will learn principles around designing TinyML model architectures, hardware-aware training strategies, effective inference optimizations, and benchmarking methodologies. This unique combination equips readers in both academic and industrial spheres with universal concepts essential for implementing TinyML in production settings.

### B. ARTICLE ORGANIZATION

The survey is organized as follows. Section II provides an extensive overview of the existing surveys on TinyML, indicating the differences with this work, while Section III describes the article selection criteria used in creating this systematic review. Section IV clarifies what it means to design a TinyML solution in terms of workflows, as we briefly explained in the previous subsection. Section V is the survey's core and presents the collection of algorithms and techniques to enable efficient ML on tiny devices. Section VI reports several state-of-the-art hardware specifications, libraries, and software platforms for TinyML application development. Section VII summarizes the overview and proposes potential future directions. Finally, Section VIII concludes the survey.

## II. EXISTING SURVEYS

In this section, we provide discussions with the most recent surveys [1], [22], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43].

### A. HARDWARE PERSPECTIVE

From the point of view of hardware technologies, we can distinguish TinyML solutions based on ***Application-Specific Integrated Circuits (ASICs)***, ***MCUs***, and ***Field Programmable Gate Arrays (FPGAs)***, in increasing order of power consumption (see Figure 1). Specific surveys exist for each one of these technologies.

*ASIC* is the focus of [42]. The RISC-V, i.e., the fifth generation of the Berkeley Reduced Instruction Set Computer (RISC) architecture, has been widely adopted by many researchers and commercial users, with several openly available implementations to choose from. Selecting the appropriate combination of RISC-V processor cores, architectures, configurations, and ML software frameworks is not trivial. To facilitate this process, the survey discusses the various RISC-V-based hardware implementations in terms of available cores and System-On-Chip (SoC) in conjunction with the software frameworks and software stacks for SoC generation. It includes a review of the latest released

frameworks supporting open hardware integration for ML applications.

*MCU* and TinyML are the subjects of [28], [33], and [34]. In [28], the authors analyze the TinyML frameworks for integrating ML algorithms within MCUs and present a real-world case study. They first give a small overview of the ecosystem of applications in which TinyML techniques can be applied and then highlight the opportunities in various sectors currently undergoing a digital transformation. Finally, they propose a Multi-Radio Access Network (Multi-RAT) architecture for smart frugal objects: i.e., sporadically messaging interconnected devices with constrained resources.

In [33], the authors survey, compare, and evaluate seven different recent and popular MCUs on a face recognition task based on a Convolutional Neural Network (ConvNet) workload. Their evaluation considers four key metrics (power efficiency, energy per inference, inference efficiency, and inference time) that can be used to benchmark ML applications on MCU-based devices.

*FPGA* is considered in [41]. The authors present ''CFU Playground'', a full-stack open-source framework that enables the rapid and iterative design of ML accelerators for embedded ML systems through Custom Function Units (CFU), i.e., hardware that augments the standard functions of a CPU. This toolchain integrates open-source software, register transfer level generators, and FPGA tools for synthesis, place, and route. To illustrate their approach, they apply their methodology to two common TinyML use cases: image classification and keyword spotting. In the first case, they show how to easily obtain iterative hardware-software improvements and, in the second, how to co-optimize the CPU and the CFU together in severely resource-constrained environments.

### B. APPLICATION PERSPECTIVE

In relation to applied TinyML, we review surveys in the fields of ***IoT***, ***environmental challenges***, ***predictive maintenance (PdM)***, ***anomaly detection***, and ***healthcare***.

In [1], the authors provide background information on the benefits that TinyML can offer to the *IoT* panorama, such as low latency, effective bandwidth utilization, strengthened data safety, and enhanced privacy. Then, they show how to implement TinyML-as-a-service, i.e., an IoT device that concretely takes part in the execution of intelligent services. In [35], the authors explore the integration of TinyML with network technologies such as 5G and LPWAN. Ultimately, we anticipate that this analysis will serve as an informational pillar for the IoT/cloud research community and pave the way for future studies.

Of particular interest in recent years, TinyML has been applied to *environmental challenges* [38] such as global warming, climate change, natural resource scarcity, and pollution monitoring. With their ability to deploy intelligent analysis together with sensing devices, TinyML provides the natural evolution to data gathering in the environmental domain to protect our societies and the natural world. This survey elaborates on the role of TinyML devices and their limit in this context.

In [32], the authors investigate techniques used to optimize TinyML-based *PdM* systems. They describe PdM and how TinyML can provide an alternative to cloud-based PdM, showing commonly used libraries, hardware, datasets, and models. Furthermore, they show known techniques for optimizing TinyML models.

*Anomaly detection* focuses on detecting abnormal behavior in the equipment by analyzing the historical data. In [29], the authors highlight the current state-of-the-art works on TinyML for anomaly detection, providing suggestions on the research direction and introducing potential future endeavors.

An essay on TinyML approaches for *healthcare* is presented in [30]. The authors collect references related to *i)* the selection of patients for investigation, monitoring, and protocol adherence, *ii)* the collection, processing, analysis, and management of data, and *iii)* drug validation trials, followed by the solutions they bring, especially using wearable devices.

Finally, in [43], the authors present an overview of many TinyML *applications* and related research efforts. Specifically, the survey builds a taxonomy of TinyML techniques that have been used so far to bring new solutions to various domains, such as healthcare, smart farming, environment, and anomaly detection.

### C. FIELD VIEWPOINT

In [27], the authors discuss the challenges and directions toward developing a fair and useful TinyML *benchmarking* suite. The group has selected four target use cases: audio wake words, visual wake words, image classification, and anomaly detection. For each use case, reference datasets and baselines were also selected. The benchmarking suite provides results regarding the model's accuracy, inference latency, and energy consumption. Notably, this is one of the few literature reviews that presents datasets that can be particularly useful for the benchmarking and design of TinyML systems.

In [31] and [37], the authors present the *background* of TinyML, list the tool for supporting TinyML, and the key enablers (e.g., model compression and quantization) for the improvement of TinyML systems. However, neither of them is focused on advanced learning aspects for TinyML, as well as established HW-SW co-design support to enhance TinyML systems, making our survey clearly distinct from these two.

In [36], the authors aggregate the key challenges reported by TinyML developers and identify state-of-art *Software Engineering (SE)* approaches that can help address key challenges in TinyML-based IoT embedded vision. These challenges include the lack of curated datasets derived from IoT-embedded vision sensors, the application portability across different devices and vendors, and the compiler choices, since embracing sophisticated compilers can help optimize for specific MCU targets. However, this affects portability, and hence, it challenges large-scale deployment under availability constraints.

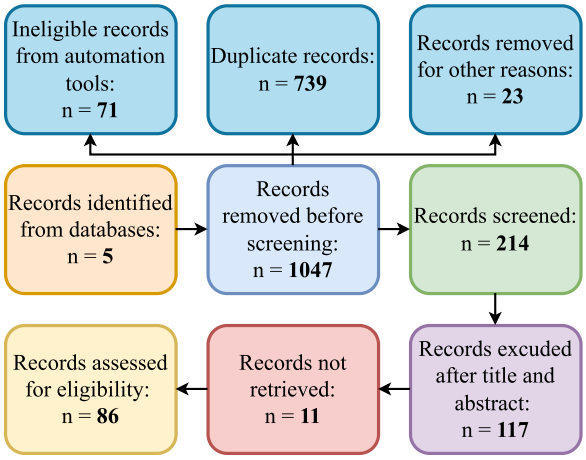



**FIGURE 3. PRISMA-based flowchart of the retrieval process. The image has been changed from the standard flowchart solely for aesthetic purposes.**

In [39], the authors focus on edge *training* and edge *inference*. This paper provides a survey of existing architectures, technologies, frameworks, and implementations in these two areas and discusses existing challenges, possible solutions, and future directions.

Finally, in [40], a review of *deployment techniques* for TinyML devices are provided, with also numerical insights to prove which deployment workflow is more promising given the constraints of the models and input data (e.g., sparsity, compression, etc.). They also inspect the engineering of reducing the computation and memory footprint for the inference of already existing models. Furthermore, they set up some case studies to present the deployment of several famous models with and without various techniques (such as compression and feature projection), with numerical results to highlight the benefits of each technique.

Most of the previously listed works assume that TinyML can only run inference on data. Despite this, growing interest in TinyML has led to work that makes them *reformable*, i.e., work that permits TinyML to learn from new data points once deployed. This originates from the need to combat model drift – the inevitable degradation of a model's performance due to the ever-changing nature of data. In [22], the authors provide a survey on reformable TinyML solutions.

It should be noted that the majority of these works are not conducted based on a well-defined and widely known Systematic Literature Review (SLR), e.g., using Preferred Reporting Items for Systematic reviews and Meta-Analyses (PRISMA) [44], except [36] and [43].

## III. SELECTION CRITERIA

This section describes the selection criteria of this systematic review, i.e., how the papers that were considered were selected.

Only publications in the English language were considered, and all studies had to be published in peer-reviewed journals or conference proceedings. The search strategy and selection criteria were developed in consultation with all authors through the Rayyan software tool for systematic literature reviews. Any disagreements between authors were resolved through discussion and consensus. To gather up-to-date knowledge from a broad spectrum of information sources, this comprehensive ML-oriented survey on TinyML was conducted following a widely known SLR methodology based on the PRISMA guidelines, the golden standard for improving transparency, accuracy, and completeness in documented systematic reviews and meta-analyses.

The included studies were extracted from the following five databases: ***Web of Science***, ***Scopus***, ***IEEE Xplore***, ***ScienceDirect***, and ***Google Scholar***, from January 2018 to January 2024. All searches included the following terms: *"TinyML"*, *"efficient machine deep learning"*, *"neural network optimization"*, *"iot machine deep learning"*, *"embedded machine deep learning"*, *"edge machine deep learning"* and *"mcu machine deep learning"*. These concepts form the basis of the inclusion criteria for selecting studies considered by the systematic review. Therefore, all the cited papers in this work were found using the above keyword combination.

Our keywords produced a total of 1,047 records. Figure 3 illustrates the PRISMA flowchart, which serves as a transparent and replicable means of reporting the systematic review's search and selection process. First, we removed all duplicate papers (739 excluded). Next, we excluded all the papers marked as ineligible by the automation tool (71 excluded) and not accessible papers (e.g., requiring paid access) (23 excluded). After the title and abstract screening process, 97 articles were selected (117 excluded). The number of records not found is 11. As a result, 86 were eligible.

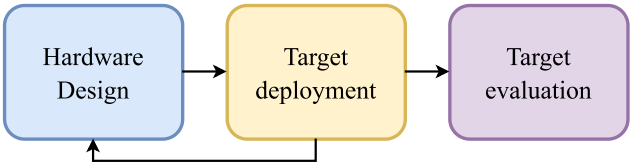
Finally, out of the 86 reviewed papers, none of them were found to be survey papers on ML-oriented techniques for TinyML. As a result, we claim that this is the first systematic review to address this topic.

## IV. TinyML WORKFLOWS

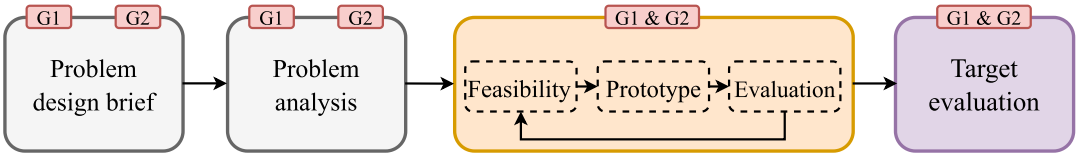
This section presents an overview of how TinyML-based systems are built. The two intrinsic ingredients of such systems are the ML model and the hardware platform. Therefore, the natural approach for developers in the field is to start working from the most familiar component. A more efficient but challenging alternative is to develop both sides from the beginning and create an integrated solution. As anticipated in Section I, the two traditional workflows for TinyML solutions are ***ML-oriented*** and ***HW-oriented***, while the holistic methodology is called ***co-design***.

In the *ML-oriented* workflow (see Figure 4.a), the majority of the expertise is in the design, adaptation, training, and evaluation of ML models, while the choice of hardware platforms is fixed or limited, due to necessity or specific industrial requirements [45], [46], [47]. A typical example of this workflow is the porting of modern neural network models to embedded devices [48]. This requires extensive experimental investigations for the implementation to be efficient in terms of power consumption, latency, and memory


(a) The **ML-oriented** workflow for implementing a TinyML-based system.




(b) The **HW-oriented** workflow for implementing a TinyML-based system.




(c) The **co-design** workflow for implementing a TinyML-based system.

**FIGURE 4.** **The three workflows for implementing a TinyML-based system: (a) ML-oriented, (b) HW-oriented, and (c) co-design. In the latter, two separate working groups, G1 (ML specialists) and G2 (HW specialists), collaborate together.**

usage, all resources in short supply on such devices compared to cloud solutions.

In particular, we identify the following stages in the ML-oriented workflow:

- **Model design**: ML practitioners formulate, train, and validate a comprehensive model suitable for the problem domain. This stage is highly dependent on the nature of this domain but disregards on purpose the specifics of the hardware platform to achieve maximum generalization and performance.
- **Model optimization**: This stage consists of different strategies to compromise performance for efficiency, discussed in more detail in Section V.
- **On-host evaluation**: The optimized model is evaluated against the performance parameters required in the specifications, and if found lacking, it is re-designed.
- **Target deployment**: Specialized optimizations are applied to the model to increase the inference efficiency by leveraging specific hardware device features.
- **Target evaluation**: The final system evaluation in production is performed.

On the other hand, in the *HW-oriented* approach (Figure 4.b), the developers are mainly focusing on designing enhanced hardware platforms that are optimized for embedded applications in order to run current and future state-of-the-art ML algorithms. This often involves investigating the bottlenecks in an existing architecture with regard to computations within an ML framework, like neural networks, and the design of hardware accelerator modules to improve throughput and consumption: e.g., reducing computational complexity in convolution layers [49], [50], efficient, low-power and feature-rich perceptrons [51], enhanced data caches [52]. In other cases, the developers design new hardware platforms optimized for embedded applications with extended digital signal processing capabilities already integrated [53]. These, in turn, require the development of optimized computing libraries [54], [55] to extract the most performances.

A HW-oriented workflow may have the following stages:

- **Hardware design**: Hardware practitioners create the design for an architecture, or accelerator module in an architecture, that improves performances for a given class of computing problems or signal processing algorithms.
- **Target deployment**: Assessment of the performance of the optimized hardware on benchmarks of the given computing problems, mostly in simulated environments. In case of unsatisfying results, return to the design stage.
- **Target evaluation**: Production and evaluation of the physical hardware devices.

Finally, in the *co-design* workflow (Figure 4.c), the approach is to integrate both sides of the development from the start to gain further improvements in performance and resource consumption. In particular, while model optimization and hardware design are separate steps in the previous workflows (Figure 4.a and Figure 4.b), here they are intertwined and co-optimized: in some cases to create bespoke architectures for specific ML algorithms on FPGAs [41], in other cases to allow neural network compu-

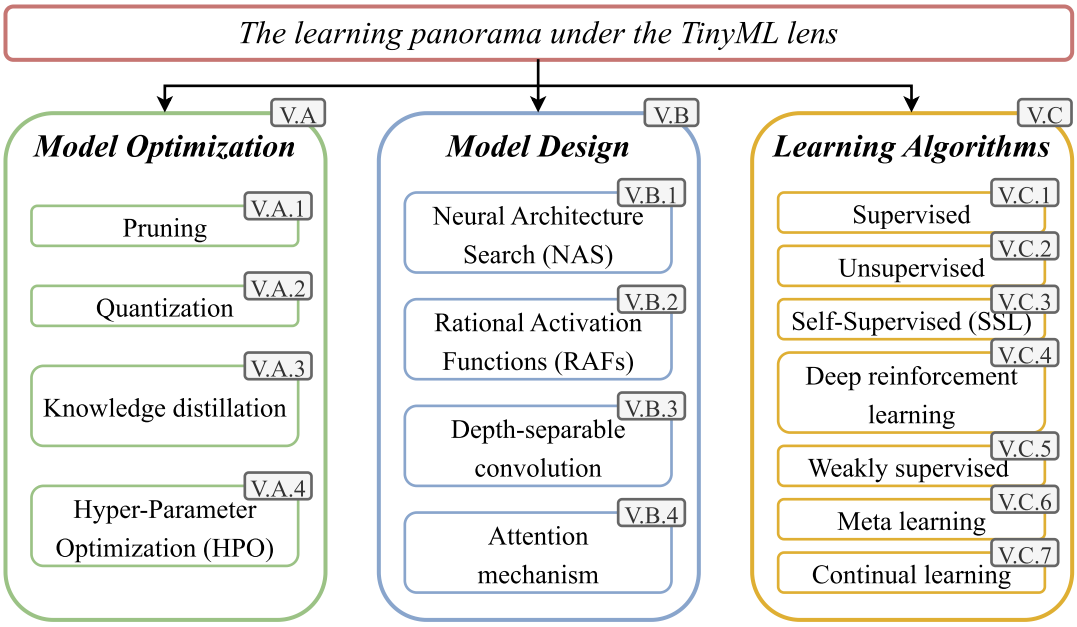



**FIGURE 5.** Our proposed taxonomy covers the learning panorama under the TinyML lens and includes three key domains: model optimization, model design, and learning algorithms.

tations on customized accelerators using analog Compute-in-Memory (CiM) hardware through HW-informed training methodologies [56].

The co-design workflow may be described with the following steps:

- **Problem design brief**: Two separate working groups, G1 (ML specialist) and G2 (HW specialist), define the capabilities and requirements of the target device.
- **Problem analysis**: The two groups specify their state-of-the-art architecture after exploring the possible alternatives.
- **Co-design (i.e., feasibility, prototype, and evaluation) step**: In a cooperative and concurrent design process, specific hardware and software components for selected sections of an application must be chosen with a global view of the system.
- **Target evaluation**: Final evaluation of the model-specific and target-specific optimizations for the device in production.

## V. LEARNING PANORAMA UNDER THE TinyML LENS

To leverage the full potential of TinyML, exploring and understanding the complexities involved in designing and optimizing ML models for specifically resource-limited devices is essential. This is true for traditional ML algorithms but is particularly relevant for approaches based on representation learning, such as those of Deep Learning (DL) algorithms. As a subset of ML, DL is increasingly used for many real-world applications, and there is a growing number of research papers on TinyML involving DL.

Thus, the learning techniques presented in the following sections are usually applied to reduce the complexity of the ML and DL algorithms so that they can run on resource-limited hardware. Since TinyML hardware has serious limitations concerning performance and every aspect of implementation, learning techniques become crucial for effectively implementing TinyML algorithms. We propose the taxonomy shown in Figure 5, which covers the learning panorama under the TinyML lens. In the following sections, we will delve into the macro-areas of ***model optimization***, ***model design***, and ***learning algorithms***.

**TABLE 1.** Popularity over the years of model optimization techniques in terms of research contributions reviewed in this survey.

| Technique | Popularity | Pre '20 | '20 - '22 | '23 |
|---|---|---|---|---|
| Pruning | High | [57]–[61] | [62]–[64] | [65], [66] |
| Quantization | High | [67]–[70] | [71]–[75] | [76]–[78] |
| Knowledge distillation | High | | [79]–[89] | |
| HPO | Low | [90]–[92] | [93] | |

### A. MODEL OPTIMIZATION

Model optimization techniques tailored for TinyML generally produce smaller memory footprints, lower energy consumption, and reduced inference latency. While ML typically requires significant computational resources, TinyML-based systems are instead limited in memory, processing power, and energy. The following paragraphs explore techniques such as ***pruning***, ***quantization***, ***knowledge distillation***, and ***Hyper-Parameter Optimization (HPO)***, that enable efficient model deployment. Table 1 provides an overview of the popularity of these techniques within the TinyML field, considering the referenced research contributions.

#### 1) PRUNING

This process eliminates weight connections within a network to accomplish different goals like reduced model footprint and accelerated inference speed [64]. Despite a lack of

standardized benchmarks and metrics due to differences in goals favoring different design choices and evaluations, pruning is effective at compressing models while keeping (or sometimes increasing) accuracy [62]. Pruning techniques can be applied during the training process or after the model has been trained. During training, pruning regularizes the model, mitigating the risk of overfitting [59]. Instead, Post-training pruning is employed to eliminate redundant connections and parameters from the model, enhancing its efficiency and accelerating its execution [61]. We identified in the current literature three main approaches: ***weight pruning***, ***neuron pruning***, and ***structured pruning***.

Specifically, *weight pruning* is a technique that eliminates connections or weights in a model falling below a given threshold for weight size [60]. Approaches based on this technique are gaining interest due to their immediate applicability [65], [66]. Similarly, *neuron pruning* discards entire neurons based on a given threshold of importance [57] and *structured pruning* removes entire structures or sub-networks from a model [58].

#### 2) QUANTIZATION

This involves performing computations and storing tensors at lower bit widths compared to floating point precision [73]. By utilizing fewer bits to represent data, such as 16-bit floats or 8-bit integers instead of 32-bit floating-point numbers, quantization enables more compact model representations and the utilization of efficient vectorized operations on various hardware platforms [69]. This technique is particularly beneficial during inference, significantly reducing computation costs while maintaining inference accuracy [67]. Quantization can be achieved through two approaches: ***Quantization-Aware Training (QAT)***, which involves re-training the model, and ***Post-Training Quantization (PTQ)***, which applies quantization without re-training.

*QAT* involves quantizing a pre-trained model and subsequently performing a fine-tuning step to recover any accuracy loss caused by quantization-related errors, which may impact model performance [74]. The QAT process consists of two stages: pre-training and fine-tuning. In the pre-training stage, the network is trained using standard techniques in a full-precision floating-point format (32-bit) to learn data patterns and develop robust feature representations. In the fine-tuning stage, the network is converted to a quantized representation, combining fixed-point and floating-point arithmetic. This adjustment allows the network to adapt to the quantized representation while preserving accuracy. QAT encompasses different methods, including *hybrid* [70], *layer-wise* [94], and *adaptive* approaches [95].

*PTQ* reduces memory usage and computational costs by converting model weights and activations from high-precision floating-point to low-precision numbers [68]. Initially, the model is trained using floating-point representation, followed by quantization of weights and activations using techniques like k-means clustering or vector quantization [71]. Adopting low-precision numbers, such as 8-bit integers, significantly reduces memory requirements, enabling more efficient model execution and suitability for resource-constrained environments [72]. These techniques are also known as *Dynamic Range Quantization (DRQ)* or *Full-Integer Quantization (FIQ)*, depending on whether only the weights or also the inputs and activation functions are being quantized to 8-bit integers.

Like in the case of pruning, the application of quantization techniques allows for immediate deployment of already existing models to resource-constrained devices in various fields like computer vision [76], [77] and healthcare [78].

#### 3) KNOWLEDGE DISTILLATION

This technique transfers knowledge from a large, complex model (teacher) to a smaller, simpler model (student) [84]. This process is important for various reasons, such as reducing computational demands or enhancing model performance on specific tasks. Knowledge types, distillation strategies, and teacher-student architectures are vital factors in student learning during knowledge distillation. The subsequent paragraphs introduce the key categories of ***knowledge types*** and ***distillation strategies***.

The extraction of *knowledge* from teachers and its utilization for training student networks can be classified into three categories: ***response-based***, ***feature-based***, and ***relation-based***. Specifically, *response-based* knowledge distillation involves mimicking the final predictions of the teacher model by capturing the neural response in the last output layer [87]. *Feature-based* knowledge expands upon this approach by using both the outputs of the last layer and intermediate layers to train thinner networks [81]. Finally, *relation-based* knowledge takes a step further by exploring the relationships between different layers or data samples in addition to the outputs of specific layers in the teacher model [86].

The *distillation* schemes are also crucial for the student learning process. Depending on the training strategy, the following three different categories are presented: ***offline distillation***, ***online distillation***, ***self-distillation***. *Offline distillation* is a two-stage strategy, where the teacher model is first trained on a set of training samples, and then the trained teacher model is used to guide the student model by extracting intermediate features or logits [80]. On the other hand, *online distillation* is an end-to-end approach where both the teacher and student models are updated simultaneously, making it suitable when the teacher model is not significantly larger or higher performing [85]. Finally, *self-distillation* is a special case of online distillation where the teacher and student networks have the same architecture [79].

In general, knowledge distillation is used to achieve a good trade-off between small model size and an acceptable accuracy [88]. For this reason, it is widely adopted in several fields where existing models are well-performing but unable to be deployed "as they are" in resource-constrained hardware. This is the case with large scaling requirements [89], bandwidth-limited domains [82], and

**TABLE 2. Popularity over the years of model design techniques in terms of research contributions reviewed in this survey.**

| Technique | Popularity | Pre '20 | '20 - '22 | '23 |
|---|---|---|---|---|
| NAS | High | | [96]–[101] | [102]–[104] |
| RAF | Low | [105] | [25], [106] | |
| Depth-separable | Medium | [107]–[109] | [110] | |
| Attention | Medium | [111], [112] | [113]–[117] | |

healthcare applications, where the trade-off between accuracy and model size needs to produce a high accuracy model that can fit the hardware requirements [83].

#### 4) HYPER-PARAMETER OPTIMIZATION (HPO)

This technique automates the search for the optimal hyper-parameter values of a model to enhance its performance on a specific task [90]. Hyper-parameters, such as learning rate, batch, and network size, are predetermined parameters that influence model behavior [92].

HPO utilizes search algorithms, such as *Grid Search*, *Random Search*, and *Bayesian Optimization*, to explore the hyper-parameter space and identify the combination that yields the best performance [91]. By automating the tuning process, HPO reduces the effort and time required while improving the model's performance.

### B. MODEL DESIGN

Unlike traditional ML models, TinyML models require careful design considerations to balance accuracy and efficiency. This section investigates techniques in model architecture exploration, model simplification, and architectural modifications that provide lightweight models capable of delivering acceptable performances for their intended applications. In the following paragraphs we explore ***Neural Architecture Search (NAS)***, ***Rational Activation Functions (RAFs)***, ***depth-separable convolution***, and the ***attention mechanism***. Table 2 provides an overview of the popularity of these techniques within the TinyML field, considering the referenced research contributions.

#### 1) NEURAL ARCHITECTURE SEARCH (NAS)

Neural architecture design plays a crucial role in data representation and performance but heavily relies on researchers' knowledge and experience. NAS automates the process of discovering optimal architectures for specific needs, replacing manual tweaking with an automated exploration of more complex architectures.

NAS utilizes search algorithms, such as ***reinforcement learning***, ***evolutionary algorithms***, and ***gradient-based methods***, to identify architectures that maximize performance on a given task. Moreover, in [102], the authors argue that it is beneficial to NAS approaches for resource-constrained systems to also search for appropriate data granularity. Specifically, data granularity refers to the concept that data can be fed into an ML model at various levels of detail (e.g., an audio sample can be presented to an ML model using different sample rates). By automating the search process, NAS reduces the time and effort required for network design and optimization, leading to improved task performance [99], [100], [101], [103].

For example, in these works [97], [98], [104], NAS algorithms targeted specifically to microcontrollers are investigated, demonstrating that NAS promises to help design accurate ML models that meet the tight MCU memory, latency, and energy constraints [96].

#### 2) RATIONAL ACTIVATION FUNCTIONS (RAFs)

Activation functions play a central role in DL since they form an essential building stone of neural networks. Thus, identifying new activation functions that can potentially improve the results is still an open field of research. Recently, RAFs have awakened interest because they were shown to perform on par with state-of-the-art activations on image classification [105]. They are trainable in an end-to-end fashion using backpropagation and can be seemingly integrated into any neural network in the same way as common activation functions (e.g., ReLU). In other words, the key idea is to involve the activation functions in the learning process together (or separately) with the other parameters of the network, such as weights and biases.

RAFs have several advantages over standard activation functions [106]. For example, they can provide better approximation capabilities, which can improve the performance of the neural network [25]. Additionally, RAFs can have more flexible shapes, making them better suited for modeling a wider range of data distributions.

Thus, by exploring RAFs, which can potentially strike a balance between accuracy and computational cost, we could unlock new avenues for creating compact yet high-performing models ideal for resource-constrained contexts. Despite the scarcity of previous research on the subject, examining RAFs might lead to ground-breaking findings and innovative insights in refining TinyML models for real-world applications.

#### 3) CONVOLUTIONAL LAYERS

In the convolution operation, each filter convolves over the input's spatial and channel dimensions. The filter size is typically denoted as $s_x \times s_y \times in_{ch}$. Standard convolutions have a high computational cost, depending on the kernel and input sizes. To optimize this process, depth-separable convolution was introduced. Specifically, depth-separable convolution involves two steps:

1) Performing a point-wise convolution with $1 \times 1$ filters, resulting in a feature map with a depth of $out_{ch}$.
2) Conducting a spatial convolution with $s_x \times s_y$ filters in the $x$ and $y$ dimensions.

By stacking these two operations without intermediate non-linear activation, the output shape remains the same as that of a regular convolution but with significantly fewer parameters.

This technique is utilized in models like MobileNet [107] and MobileNetV2 [109], designed for mobile and embedded devices. Using depth-wise separable layers instead of regular convolutions, MobileNet reduces the number of parameters and multiply-add operations, enabling efficient deployment on mobile devices for computer vision tasks.

Furthermore, in [110] Tiny-Sepformer is presented, a tiny time-domain transformer network that uses Convolution-Attention (CA) block into the masking network to split the layer into convolution path and attention path parallelly. In particular, to further reduce the computation, the convolution part of CA is a 1D depthwise separable convolution.

Finally, in [108] Xception is presented, an interpretation that considers Inception modules in convolutional neural networks as an intermediate step between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution).

#### 4) ATTENTION MECHANISM

While commonly associated with notable contributions to machine translation tasks, this technique has been adapted and adopted for a wide range of applications [115]. Its fundamental purpose remains unchanged: allow the model to focus on relevant input parts while generating outputs. This enables the model to selectively attend to different regions or features, thereby facilitating the extraction of salient information from complex and high-dimensional data. Beyond translation tasks, the attention mechanism has been successfully employed in natural language processing [113], image captioning [112], speech recognition [111], and more. By incorporating attention into these tasks, models can effectively handle long-range dependencies, capture context-specific information, and improve overall accuracy and robustness.

The beauty of the attention mechanism lies in its ability to dynamically assign importance to different parts of the input based on their relevance to the current context. This adaptability allows models to prioritize relevant information and disregard noise or irrelevant details, resulting in more precise and context-aware predictions. Hence, attention can be particularly useful in TinyML applications, where resource-constrained devices require efficient and compact models. In [114], the authors introduce AttendNets, a deep self-attention architecture based on visual attention condensers, to deploy on-device visual perception tasks like image recognition.

### C. LEARNING ALGORITHMS

Among the many taxonomies covering the whole discipline of ML, we focus first on the standard paradigms of ***supervised***, ***unsupervised***, ***Self-Supervised (SSL)***, and ***deep reinforcement learning***. Moreover, for the particular case of optimal usage of resources, we review ***weakly-supervised learning***, ***meta-learning*** and ***continual learning*** techniques, detailing how these are useful for TinyML. Also in this case, Table 3 provides an overview of the popularity of these techniques within the TinyML field, considering the referenced research contributions. Moreover, for each technique, it also examines its application across various domains. This analysis aims to underscore the most promising approaches within each application domain.

#### 1) SUPERVISED LEARNING

In this paradigm, a model learns from labeled training data to make predictions. It involves training a model on input-output pairs, where the input data is fed into the model, and the corresponding desired output or label is provided. This training process allows for robust models but at the cost of requiring a large number of annotated data. Supervised learning generally guarantees good performances when a lot of training data is available, which usually also requires big models with a high number of parameters capable of learning the patterns present in the data.

In this sense, there are TinyML approaches that perform supervised learning on big models and then distill the knowledge on small models, leveraging the knowledge learned from the big supervised models. This can be performed by compressing a previously trained model (teacher) to a smaller one (student) as in [118], in a process that is similar to the knowledge distillation process (as explained in Section V-A3), or by learning directly from a big model leveraging parallel computation, as proposed by [119].

As stated before, supervised learning requires a lot of annotated data to be effective. Despite that, some TinyML works try to compensate for the lack of annotation on the data. In this regard, in [120], the authors implement a method for generating missing samples during training in the context of human action recognition, where missing samples could lead to inaccurate classifications. In [121], they propose to take into account the uncertainty of future samples in a power consumption management implant. This is performed by trying to capture the underlying seasonal and daily changes from some annotated data and then forecast the uncertainty of future energy consumption.

#### 2) UNSUPERVISED LEARNING

In this paradigm, a model learns from unlabeled data, striving to discover patterns, structures, or relationships in the data.

Anomaly detection is one of the most common use cases for the unsupervised learning approach. This makes anomaly detection particularly interesting for TinyML, given the need to process raw data streams as close to their origins in their early stages. Specifically, the industrial environment is where most of the research and development related to anomaly detection is concentrated [123]. This is due to the unique challenges encountered in such environments, including limited or unreliable communication with the cloud, uninterrupted connectivity, and potential obstacles to accessing systems. In this sense, TinyML becomes of necessary importance.

Besides the industrial environment, other real-world scenarios are interested in unsupervised anomaly detection, such

**TABLE 3. Popularity over the years of learning algorithms in terms of research contributions reviewed in this survey. For each technique, the distribution of application domains highlights the most common use cases.**

| Technique | Popularity | Pre '20 | '20 - '22 | '23 | Application Domains |
|---|---|---|---|---|---|
| Supervised | Medium | | [118], [119] | [120], [121] | Classification (50%)<br>Autonomous Driving (25%)<br>Regression (25%) |
| Unsupervised | High | | [122]–[127] | | Anomaly Detection (100%) |
| SSL | Low | | [128], [129] | [130] | Anomaly Detection (100%) |
| Deep RL | Medium | [131] | [132]–[134] | [135] | Resource Optimization (66%)<br>Path Planning (33%) |
| Weakly supervised | Low | | [136] | [137] | Anomaly Detection (100%) |
| Meta learning | Medium | [138] | [139]–[142] | [47], [143] | Classification (80%)<br>Path Planning (20%) |
| Continual learning | High | [144] | [145]–[149] | [150] | Classification (86%)<br>Anomaly Detection (15%) |

as physiological disorders [122] and climate conditions [124], in which classic machine and DL models are proposed.

Specifically regarding DL approaches, in [125], a model based on autoencoders has been engineered to be executed on a microcontroller for detecting anomalies of top-load industrial washing machines. The model has been ported to an Arduino Nano microcontroller, achieving high accuracy and recall performances with remarkably low power usage.

Finally, related to the empirical data analysis approach, in [126], the authors propose an unsupervised TinyML approach to detect anomalies on roads based on the concept of Typicality and Eccentricity of Data (TEDA). Similar work is presented in [127], where the focus lies on monitoring the release of greenhouse gases from urban vehicles. Specifically, a TinyML unsupervised methodology is employed to quantify $CO_2$ emissions for the evaluation of air quality within urban environments.

### 3) SELF-SUPERVISED LEARNING (SSL)

Supervised learning is currently facing a bottleneck due to its significant dependence on expensive manual labeling, leading to issues such as generalization errors and spurious correlations [128]. SSL has emerged as a highly promising technique to address the aforementioned challenges, offering a solution that eliminates the need for costly and expensive manual annotations [130].

In SSL, the model is trained to predict certain aspects of the input data without relying on external annotations. This can be achieved by creating surrogate tasks such as, in the case of image-based tasks, predicting missing parts of an image, reconstructing an image from a corrupted version, and predicting the relative position of image patches. In this manner, the model is encouraged to capture the underlying structure and semantics of the data.

The synergy between SSL and anomaly detection is evident: by reconstructing or predicting parts of the data without explicit labels, the model learns to capture the inherent structure of the majority class, making it more sensitive to deviations and anomalies. Furthermore, it is particularly advantageous when labeled anomaly data is scarce or expensive to obtain, as it enables the model to generalize better and identify anomalies effectively in diverse and complex datasets. Since anomaly detection faces a scarcity of labeled anomaly examples, SSL leverages unlabeled data to learn features that capture the underlying structure of the data, effectively utilizing the abundance of unlabeled data available.

Specifically, in [129], an SSL method for anomaly detection on IoT devices is proposed. Their method is based on a multivariate Long Short-Term Memory (LSTM) autoencoder. The self-supervision is performed by detecting data that significantly deviates from the learned distribution and using them as anomalous data to enhance the detection of such anomaly types.

### 4) DEEP REINFORCEMENT LEARNING

Reinforcement Learning (RL) is an ML technique that trains a model (usually an agent) to take actions (policies) based on the input. The way the model learns is usually based on rewards assigned to a good policy that the model needs to maximize in the form of a Markov Decision Process (MDP) [131]. Integrated with DL, i.e., deep reinforcement learning, the optimal policies are efficiently obtained. This is useful because, in real-world applications, the space state is high-dimensional, and the use of traditional RL algorithms is not effective [151]. Specifically, in the case of TinyML, the challenge is to embed the neural networks implemented with deep reinforcement learning approaches on small, constrained devices such as MCUs.

Transferring trained deep reinforcement learning models on constrained devices is possible using several general-purpose techniques usually designed to alleviate the system resource bottlenecks, as proposed by [132] in their framework suite, making deep reinforcement learning feasible for TinyML platforms. Moreover, in [133], the authors propose the framework TinyRL to transfer the

deep reinforcement learning knowledge into resource-limited devices.

Cheap off-the-shelf MCU devices are particularly interesting for deep reinforcement learning as they are widely adopted in robotics. Deploying a DRL model on an MCU-powered intelligent agent for autonomous driving, for instance, is what the authors of [134] propose to achieve. They present a deterministic policy gradient algorithm that takes into consideration the computation energy and caching costs jointly. This significantly reduces the energy cost of the final model. Moreover, in the efforts of allowing deep reinforcement learning on MCU, [135] proposes to train a tiny ConvNet that can be easily deployed on an MCU, with the aim of solving a physical, electrically actuated tilting maze with repositionable walls.

#### 5) WEAKLY SUPERVISED LEARNING

This paradigm lies between supervised and unsupervised learning. It involves training a model using partially labeled or noisy labeled data, with only limited or incomplete supervision. Since collecting large amounts of accurately labeled data can be expensive and time-consuming, weakly supervised learning allows for training models with fewer labeled samples, reducing the labeling costs associated with data collection. This cost-efficiency is clearly advantageous for TinyML applications, where resource constraints often limit the availability of labeled training data.

For example, in [136], the authors propose a weakly supervised learning solution for improving anomaly detection performances. In particular, the training phase of the model is improved by some labels in the dataset: in fact, a part of the dataset is labeled, playing the role of ''domain expert'', which allows weakly supervised learning. The ML model used is an Isolation Forest, and these labels are used to remove unnecessary trees and keep the most informative ones, i.e., those that give the best results.

In [137], another TinyML-based system for anomaly detection in industrial environments is presented. In this case, an ensemble of ML classifiers detects if a sample is anomalous or not. This allows the system to be scalable w.r.t. the size of the ensemble, with a predictable impact on the memory footprint and delay in inference mode.

#### 6) META LEARNING

Unlike traditional methods that solve tasks independently using a fixed learning algorithm, meta-learning enhances the learning algorithm itself based on experiences from multiple learning episodes [24]. While various perspectives on meta-learning exist, our focus here is on the optimization approach known as neural-network meta-learning, which is particularly relevant for TinyML applications. The neural-network meta-learning design should take into account three independent axes that represent the current meta-learning landscape: ***meta-representation***, ***meta-optimizer***, and ***meta-objective***.

In particular, the concept of *meta-representation* [138] involves learning a high-level representation that captures the commonalities and patterns across different tasks. This meta-representation is a knowledge base from which the model can quickly adapt to new tasks with limited data. For example, in [143] MetaLDC is proposed, a system that meta-trains ultra-efficient low-dimensional computing classifiers to enable fast adaptation on tiny devices with minimal computational costs. Specifically, during the meta-training stage, MetaLDC meta-trains a representation offline by explicitly taking into account that the final (binary) class layer will be fine-tuned for fast adaptation for unseen tasks on tiny devices; during the meta-testing stage, MetaLDC uses closed-form gradients of the loss function to enable fast adaptation of the class layer.

On the other hand, the *meta-optimizer* [139] learns to optimize the model's parameters to facilitate fast adaptation and generalization across tasks. Specifically, the meta-optimizer tunes the learning algorithm itself, enabling it to update the model based on task-specific information efficiently. TinyReptile, a simple but efficient meta-optimizer-based algorithm to collaboratively learn a solid initialization for a neural network across tiny devices, is presented in [47].

Lastly, the *meta-objective* [140] guides the meta-learning process by defining a criterion for evaluating the performance of the meta-learner. It provides a signal for learning to adapt and generalize, encouraging the acquisition of task-agnostic knowledge that can be applied to new tasks. To the best of our knowledge, the two closest works in this area are [141], in which the authors propose an Adaptation-aware Network Pruning (ANP), a novel pruning scheme that works with existing meta-learning methods for a compact network capable of fast adaptation, and [142], in which it is shown that the application of Lottery Ticket Hypothesis (LTH) to meta-learning enables the adaptation of meta-trained networks on various IoT devices.

#### 7) CONTINUAL LEARNING

This research field aims to develop algorithms that enable models to continuously learn from new data while preserving previously acquired knowledge. This is essential because conventional ML models typically struggle to learn from new data while retaining previously acquired knowledge, often leading to catastrophic forgetting [144].

Recently, there has been a significant development in the field, with several promising algorithms and architectures being proposed and showing improved performance in various continual learning benchmarks. These approaches can be classified into the following three categories: ***regularization-based***, ***replay-based***, and ***dynamic architectures***.

Specifically, *regularization-based* methods introduce regularization terms into the loss function to encourage the model to maintain its prior knowledge while learning new tasks. On the other hand, *replay-based* methods involve storing past data and replaying it during training to prevent forgetting.

Finally, *dynamic architectures* adjust their capacity to accommodate new information.

The significance of this research area has increased due to the surging demand for TinyML models in several applications, including healthcare, wearables, and IoT devices. Recent research in this domain has focused on developing efficient algorithms that can manage the restrictions of TinyML devices, such as limited memory and processing power. One of the promising methods in a TinyML scenario is to use regularization-based approaches that add penalties to the loss function to prevent overfitting and catastrophic forgetting. Another effective approach is to use dynamic architectures that can adapt their structure to accommodate new tasks. For instance, in [149], a regularization-based approach for an IoT scenario is presented, in which MCUs are exploited as edge devices for data processing considering two tasks: gesture recognition based on accelerometer data and image classification.

Replay-based methods, on the other hand, are well adopted in real-world scenarios as the general replay approach is very intuitive. In [147], the authors leverage the quantization of the frozen stage of the model, allowing for 8-bit execution and replays in the latent space to reduce their memory cost with minimal impact on accuracy. The results show that by combining these techniques, continual learning can be achieved in practice using less than 64MB of memory, an amount compatible with embedding in TinyML devices. In [148], the authors propose Train++, an incremental replay-based training algorithm that trains ML models locally at the device level (e.g., on MCUs) using the full n-samples of high-dimensional data. Train++ enables resource-constrained MCU-based IoT edge devices to locally build their own knowledge base on the fly using the live data, thus creating smart self-learning and autonomous problem-solving devices. The authors of [150] propose TyBox, a toolbox for the automatic design of on-device TinyML classification models, with the idea of automatically generating the ''incremental'' version of an initial (static) pre-trained model using replays.

Lastly, regarding the dynamic architectures approaches for continual learning, in [145], a pioneering contribution in the form of Tiny-Transfer-Learning (TinyTL) is presented. In their work, the authors propose a novel approach that achieves memory efficiency by selectively freezing the network weights while solely focusing on learning the bias modules, thereby obviating the need to store intermediate activations. A new memory-efficient bias module, referred to as the lite residual module, is introduced to ensure the model's adaptability. Through extensive experimentation, it is demonstrated that TinyTL yields substantial memory savings with minimal sacrifice in accuracy compared to the conventional fine-tuning approach applied to the entire network. Finally, in [146], the authors propose TinyOL (TinyML with Online Learning), which enables incremental on-device training with streaming data.

## VI. TinyML DEVICES AND TOOLS

TinyML heavily depends on hardware devices to enable efficient training and inference for its applications. Based on our literature research [28], [33], [33], [34], [41], [42], in this section, we will examine processors for TinyML workloads spanning from general-purpose ***Central Processing Units (CPUs)*** to more programmable and adaptable architectures with discussions on ***Graphics Processing Units (GPUs)***, ***FPGAs***, and ***Tensor Processing Units (TPUs)***. By structuring the analysis along this range, we aim to illustrate the fundamental trade-offs between efficiency, programmability, and flexibility. The optimal balance point depends on the constraints and requirements of the target application. This perspective provides a framework for reasoning about hardware choices for ML and the capabilities required at each level of specialization. Table 4 compares these different hardware devices, outlining their advantages and disadvantages.

### A. CENTRAL PROCESSING UNIT (CPU)

The primary objective of TinyML is to optimize ML workloads to allow them to be executed on microprocessors with extremely low power consumption, often just a few milliwatts. Microprocessors, particularly the Arm Cortex-M family, are an ideal platform for implementing ML due to their widespread usage [152]. This versatility is due to their standardized instruction sets and mature compiler ecosystems. Additionally, their minimal power requirements make them suitable for deployment in environments where replacing batteries is challenging or inconvenient [153].

However, despite their advantages, microprocessors exhibit several drawbacks. Their wide-ranging applicability leads to the inclusion of unnecessary operations and logic checks, which might degrade computational performance. In addition, this fails to fully exploit the potential parallelism offered by DL algorithms.

### B. GRAPHICS PROCESSING UNIT (GPU)

Originally designed for accelerating computer graphics, GPUs differ from CPUs in their composition. While CPUs consist of a few Arithmetic Logic Units (ALUs) optimized for sequential processing, GPUs are equipped with thousands of ALUs that enable parallel execution of numerous simple operations. This architecture makes GPUs highly suitable for ML tasks since they can rapidly perform many parallel computations. For example, ML algorithms often involve extensive matrix and vector operations, which can be efficiently parallelized and executed on GPUs, as we can see in [154]. In particular, modern GPUs have evolved to include specialized hardware support for essential AI operations, such as Generalized Matrix Multiplication (GEMM), native support for quantization, and native support for pruning.

In recent years, NVIDIA has introduced multiple generations of GPU microarchitectures, such as the NVIDIA Jetson family, with a growing emphasis on enhancing DL

**TABLE 4. Comparison between the different hardware devices for a TinyML-based system.**

| Hardware | Advantage | Disavantage |
|---|---|---|
| CPU | Fit for general purpose, high memory capacity | Low parallelism, low throughput performance |
| GPU | High throughput performance, a good fit for SOTA architectures | Expensive, energy-hungry |
| FPGA | Energy efficient, flexible | Extremely difficult to use, lack of libraries |
| TPU | Potential to significantly boost inference performance | Expensive, hard to develop |

performance. Additionally, NVIDIA has introduced Tensor Cores [155], specialized execution units within their GPUs specifically designed for DL applications.

Furthermore, GPUs are specifically designed to efficiently handle large datasets and facilitate rapid data transfer between the main system memory and processing units, a significant attribute since ML typically operates on extensive real-time data [156]. Hence, the combination of parallel computing capabilities, many cores, and high-bandwidth memory access collectively establish GPU microarchitecture as a good choice for TinyML-based applications.

### C. FIELD-PROGRAMMABLE GATE ARRAY (FPGA)

An FPGA offers a high-performance, efficient, and scalable solution that can be reconfigured for different applications. Their customizable nature provides advantages for handling the intricate mathematical computations demanded by ML [157]. The key advantage of FPGAs is the ability to reconfigure the underlying fabric to implement custom architectures optimized for different models.

The fundamental building block of an FPGA's architecture is the fabric layer, comprising Configurable Logic Blocks (CLBs) and programmable interconnects. CLBs can flexibly be configured by users to perform various digital functions, including the complex mathematical operations essential for ML algorithms. Through programmable interconnects, CLBs can be interconnected in different configurations, enabling customization to suit diverse ML applications [158].

Aside from the fabric layer, an FPGA designed for ML often incorporates additional specialized hardware blocks, such as Digital Signal Processing (DSP) blocks and high-performance memory blocks. DSP blocks enhance the execution speed of intricate mathematical operations like convolutions and dot products, which are commonly employed in ML models. High-performance memory blocks also facilitate rapid access to the extensive datasets [159].

To summarize, an FPGA aims to deliver high performance, efficiency, and scalability, catering to the complex computational requirements of ML tasks. Consequently, FPGAs emerge as an ideal choice for a broad spectrum of applications in the TinyML domain.

### D. TENSOR PROCESSING UNIT (TPU)

A specialized processor, known as the TPU, has been developed by Google explicitly for ML tasks, with a specific emphasis on tensor operations [160]. TPUs comprise several parts, including a high-bandwidth memory system, a systolic array of processing units, and an interconnected network that facilitates communication between these components.

The systolic array represents the core of the TPU, which is responsible for executing tensor operations. It consists of numerous processing elements arranged in a two-dimensional grid, each interconnected to neighboring ones. This arrangement enables efficient communication between processing elements, facilitating the parallel execution of complex tensor operations. The high-bandwidth memory system ensures swift access to data necessary for tensor operations. Finally, the interconnect network links the TPU with other system components, such as the host processor and other TPUs, promoting efficient communication and coordination. Google has released Edge TPUs using the Coral platform in various form factors, ranging from a Raspberry-Pi-like Dev Board to stand-alone solderable modules [161].

In summary, the architecture of an Edge TPU is specifically designed to provide high performance, efficiency, and scalability in handling tensor operations, making it an excellent choice for TinyML applications.

### E. SOFTWARE TOOLS

As the demand for implementing ML on various hardware devices continues to grow, the software layer emerges as one of the essential components in developing TinyML-based systems. To date, prevalent frameworks heavily rely on vendor-specific operator libraries, demonstrating the significant potential for driving advancements in TinyML research. Below, we provide an overview of the main frameworks utilized in this domain:

- **TensorFlow Lite Micro** [162]: Is an open-source framework that empowers microcontrollers and similar devices with limited memory capacity to execute ML models. It operates efficiently without relying on an operating system, standard C (or C++ libraries), or dynamic memory allocation. Developed in C++11, this framework necessitates a 32-bit platform and exhibits compatibility with most Arm Cortex-M Series processors.
- **uTensor** [163]: Is a remarkably lightweight, open-source framework for ML inference. It is built upon TensorFlow and meticulously optimized for Arm targets. By converting ML models into readable and self-contained C++ source files, uTensor greatly simplifies integration with embedded projects.
- **Edge Impulse** [164]: Is a service that facilitates the development of TinyML models specifically tailored

for edge devices. The training takes place on a cloud platform, and the resulting trained model can be easily exported to an edge device. Additionally, Edge Impulse simplifies the collection of actual sensor data, enables live signal processing from raw data to neural networks, and streamlines testing procedures.

- **Embedded Learning Library** [165]: The Microsoft Embedded Learning Library (ELL) empowers users to design and implement intelligent ML models on resource-constrained platforms. Conceptually, the ELL can be seen as a cross-compiler for intelligence embedding, where the compiler operates on the laptop and generates machine code that can be executed on the embedded device.
- **X-CUBE-AI** [166]: Is an STM32Cube expansion package. Allows an automatic conversion of pre-trained artificial intelligence algorithms, including neural networks and classical ML models, for STM products. It also integrates an optimized library for STM32 ARM Cortex M-based boards.
- **uTVM** [167]: Is a compiler that offers graph-level and operator-level optimizations, enabling DL workloads to achieve performance portability across a wide range of hardware back-ends. It addresses optimization challenges specific to DL, including high-level operator fusion, mapping to various hardware primitives, and effectively mitigating memory latency.
- **MinUn** [168]: Is a framework jointly developed by Microsoft Research in India, ETH Zurich, and UC Berkeley, designed explicitly for TinyML applications. It presents a comprehensive solution to three critical sub-problems. Firstly, it addresses the challenge of utilizing number representations that approximate 32-bit floating point numbers using fewer bits without compromising accuracy. Secondly, it offers heuristic techniques to optimize bandwidth assignment, ensuring minimal memory usage while preserving accuracy. Lastly, it tackles the memory management issue on devices with limited resources, mitigating potential problems related to memory fragmentation.

## VII. DISCUSSION

In the recent five years, as shown by our search strategy, there has been a notable surge in studies investigating TinyML methods, optimizations, and applications. This trend reflects the growing recognition of the importance of real-time solutions for many complex and safety-critical real-world applications. This paper presents a comprehensive analysis from the ML point of view of TinyML. We aim to provide not just an updated guide on the current state-of-the-art but also to pinpoint areas that have yet to be explored. By doing so, we hope to lay the foundation for future research and investigations in this field.

From the proposed taxonomy in Figure 5, the area of model optimization, based on referenced research contributions, is the one that has received the most extensive exploration. Indeed, within TinyML, we come across several cutting-edge works that explore techniques such as pruning [63], quantization [75], and knowledge distillation [88]. On the contrary, the scarcity of research focusing on HPO [93] can be attributed to its complexity, the lack of awareness about the importance of HPO and its potential to enhance the performance of ML models significantly, and the resource requirements, which can be a limiting factor for researchers with restricted access to high-performance computing infrastructure. Table 1 summarizes this inquiry.

Regarding the model design area, most of the work is focused on NAS [97]. Secondly, it's worth noting that another significant portion of the existing research in this area is related to attention mechanisms [116], [117] and depth-separable convolutions since these techniques enhance model efficiency, accuracy, and real-time inference capabilities, making them essential for resource-constrained edge devices. Therefore, we expect research in these areas to grow significantly in the coming years. However, despite the numerous advancements made in ML by applying RAFs or VeLO, no such method has been found in any works related to TinyML. We believe that pursuing research in this direction could potentially lead to further enhancements in the quality of the produced models, and Table 2 summarizes this information.

Related to the learning algorithms, we have encountered a significant body of research focused on unsupervised and continual learning. Additionally, considerable efforts have been made in supervised learning, meta-learning, and deep reinforcement learning. However, we note that self-supervised and weakly-supervised learning fields still require effort before they can be widely used in TinyML-based works. Hence, we firmly believe that directing research efforts toward these two areas holds immense potential for significant advancements in the years to come. Unsurprisingly, following the trend in the ML community at large, many works use DL models to tackle real-world problems. Table 3 sums up this direction.

In the field of TinyML applications, most efforts focus on addressing the challenges of anomaly detection. This is consistent with the previous statement emphasizing the use of unsupervised and continual learning strategies. It is worth noting that within the TinyML landscape, most methodologies used to tackle various tasks lean toward DL paradigms. Non-DL algorithms are used sporadically in this context, with notable exceptions being approaches based on the TEDA framework, as demonstrated in [126].

Regarding hardware choices, there isn't a one-size-fits-all solution among CPUs, GPUs, FPGAs, and TPUs. Therefore, selecting the most suitable hardware for a specific application is essential. Each comes with its own advantages and disadvantages, as summarized in Table 4. Specifically, Table 5 reports TinyML off-the-shelf hardware and the range of devices adopted by the different techniques. This table shows a clear pattern in adopting specific low-power MCU devices, i.e., Arduino Nano 33 BLE (and its variants), and

**TABLE 5.** TinyML off-the-shelf hardware and their usages in current literature. The devices are in order of popularity (decreasing). (*) unavailable values due to missing information in the original works or derived by the board producer datasheet.

| Device | Type | CPU | Clock | SRAM | '20 - '22 | '23 | Applications |
|---|---|---|---|---|---|---|---|
| Arduino Nano 33 | MCU | ARM Cortex M4 | 64 Mhz | 256 KB | [124]–[126], [133], [146] | [47], [150] | Anomaly Detection (70%), Image Classification (30%) |
| STM32 family | MCU | ARM Cortex M | 80 Mhz - 480 Mhz | 128 KB - 192 KB | [119], [129], [147]–[149] | [135] | Anomaly Detection (65%), Autonomous Driving (25%), Image Classification (10%) |
| ESP32-based microcontrollers | MCU | Xtensa LX6 | 240 Mhz | 520 KB | [127], [133], [148] | [137] | Anomaly Detection |
| Raspberry Pi Pico | MCU | ARM Cortex M0 | 133 Mhz | 264 KB | [133] | | Optimization Problems |
| TI-CC2652 | MCU | ARM Cortex M4F | 48 Mhz | 88 KB | | [121] | Regression |
| mbed LPC11U24 | MCU | ARM Cortex M0 | 48 Mhz | 8 KB | [132] | | Optimization Problems |
| *Unspecified* | MCU | GAP8 (Risc-V) | (*) | (*) | [119] | | Autonomous Driving |
| Odroid-XU3 | CPU | ARM Cortex A15 | 2000 Mhz | 4 GB | | [120] | Action Classification |
| *Unspecified* | CPU | ARM Cortex A72 | (*) 2200 Mhz | 4 GB | [123] | | Anomaly Detection |

the STM32 family of microcontrollers. However, we expect that the co-design approach will significantly focus further advancements in this research field since the early work seems extremely promising, making it the primary direction of future developments.

RISC-V has gained significant attention recently due to its versatility and scalability. We believe that RISC-V will significantly impact future TinyML research in several areas related to hardware and software development [42]. One of the key advantages of RISC-V in the realm of TinyML is its open nature, allowing researchers and developers full access to the architecture specifications and offering a more flexible and optimized solution than proprietary architectures. Furthermore, the architecture of RISC-V makes it easier to include custom instructions designed for ML workloads, like extending the instruction set for efficient neural network inference. Thus, researchers can selectively include only the necessary instructions, reducing the overall complexity of the processor and minimizing the memory footprint. In addition to hardware implications, RISC-V significantly impacts TinyML's software ecosystem. The availability of open-source toolchains and compilers for RISC-V simplifies the development process for TinyML applications.

Aligning ML with human desiderata and ethics aims to make ML systems embody human principles. "AI for Good" initiatives support AI development for Sustainable Development Goals (SDGs). Although ML is crucial in advancing the SDGs, its adoption is hindered by high energy consumption, connectivity requirements, and cloud deployment costs. In this regard, TinyML presents a tremendous opportunity to harness the power of ML to advance the SDGs and drive social impact globally. Using TinyML, we can circumvent barriers like poor infrastructure, limited connectivity, and high costs that often exclude developing communities from emerging technology.

Despite the promising applications and growing scientific literature in the field of TinyML, further research is needed to fully comprehend its advantages and limitations. In this context, we draw other additional unresolved issues that require dedicated research to drive future advancements in the field. In particular:

- **Benchmarking**: The lack of a recognized benchmark, due to the challenges posed by low power, limited memory, hardware heterogeneity, and software heterogeneity, is an important impediment that may hamper TinyML services [27]. In this context, the IoT community has shown an increasing interest in benchmarking as a way to scientifically compare the performance of various TinyML solutions, both for training benchmark [169], for inference benchmark [170], and specifically for TinyML systems [171].
- **Memory Constraints**: The insatiable demand for computation and high accuracy has continued to push innovations in ML algorithms. However, the extremely small size of SRAM and flash memory makes the task of DL on edge devices very challenging today.
- **Data-driven engineering**: Understanding data quality thoroughly is critical because relying solely on accuracy can be misleading when predicting model behavior. We will need a large amount of relevant real-world data to accomplish this. This information will assist us in identifying specific instances where the model fails to detect or behaves incorrectly. Furthermore, post-processing techniques will be required to improve the model's performance in these areas. In essence, we need tools and processes that prioritize "data excellence" to assess data quality comprehensively.
- **Lack of accepted models**: Many DL models are widely accepted for conventional infrastructure. For example, MobileNet is the baseline for benchmarking deep neural networks in mobile edge computing devices. However,

no such popular model can be adopted for the TinyML on the MCUs ecosystem.

- **Lack of public datasets**: Despite some datasets specifically designed for TinyML being available (such as for on-device online training [148]), to date, TinyML is mainly concerned with sensor processing in general, so the question that emerges is… *"What's the ImageNet [172] of TinyML"?*

## VIII. CONCLUSION

The prodigious amount of research invested over the past decades in improving embedded technologies to enable the use of real-time solutions for many complex and safety-critical applications led to the birth of TinyML (Section I). As summarized in Figure 3, this paper presents a systematic review of TinyML from January 2018 to January 2024 (Section III). For the first time ever, we formalize the three different workflows to implement a TinyML-based system (Section IV). As an additional and distinct contribution, this survey strongly emphasizes the ML perspective. It not only presents the most current TinyML frameworks but also recommends recent variations and advancements in ML technologies that TinyML practitioners may consider exploring to enhance the state-of-the-art capabilities (Section V). In Section VI, we examine the advantages and disadvantages of different hardware devices that can be used to develop TinyML-based applications. Finally, Section VII highlights the fields that hold the most promise for further research in the upcoming years. Additionally, we provide a list of unresolved problems that must be addressed to propel the field forward.

## REFERENCES

[1] D. L. Dutta and S. Bharali, "TinyML meets IoT: A comprehensive survey," *Internet Things*, vol. 16, Dec. 2021, Art. no. 100461.

[2] M. Capra, R. Peloso, G. Masera, M. R. Roch, and M. Martina, "Edge computing: A survey on the hardware requirements in the Internet of Things world," *Future Internet*, vol. 11, no. 4, p. 100, Apr. 2019.

[3] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of Things (IoT): A literature review," *J. Comput. Commun.*, vol. 3, no. 5, p. 164, 2015.

[4] A. Daissaoui, A. Boulmakoul, L. Karim, and A. Lbath, "IoT and big data analytics for smart buildings: A survey," *Proc. Comput. Sci.*, vol. 170, pp. 161–168, 2020.

[5] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-khah, and P. Siano, "Iot-based smart cities: A survey," in *Proc. IEEE 16th Int. Conf. Environ. Electr. Eng.*, Jun. 2016, pp. 1–6.

[6] Y. Kabalci, "A survey on smart metering and smart grid communication," *Renew. Sustain. Energy Rev.*, vol. 57, pp. 302–318, May 2016.

[7] B. B. Sinha and R. Dhanalakshm, "Recent advancements and challenges of Internet of Things in smart agriculture: A survey," *Future Gener. Comput. Syst.*, vol. 126, no. 4, pp. 169–184, Jan. 2022.

[8] N. Gondchawar and R. Kawitkar, "IoT based smart agriculture," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 5, no. 6, pp. 838–842, 2016.

[9] F. Alshehri and G. Muhammad, "A comprehensive survey of the Internet of Things (IoT) and AI-based smart healthcare," *IEEE Access*, vol. 9, pp. 3660–3678, 2021.

[10] Y. Song, F. R. Yu, L. Zhou, X. Yang, and Z. He, "Applications of the Internet of Things (IoT) in smart logistics: A comprehensive survey," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4250–4274, Mar. 2021.

[11] A. Jayaram, "Smart retail 4.0 IoT consumer retailer model for retail intelligence and strategic marketing of in-store products," in *Proc. 17th Int. Bus. Horizon-INBUSH ERA*, Noida, India, vol. 9, 2017.

[12] Y. Liao, E. de Freitas Rocha Loures, and F. Deschamps, "Industrial Internet of Things: A systematic literature review and insights," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4515–4525, Dec. 2018.

[13] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[14] E. Hozdić, "Smart factory for industry 4.0: A review," *Int. J. Modern Manuf. Technol.*, vol. 7, no. 1, pp. 28–35, 2015.

[15] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[16] W. Zhang, D. Yang, and H. Wang, "Data-driven methods for predictive maintenance of industrial equipment: A survey," *IEEE Syst. J.*, vol. 13, no. 3, pp. 2213–2227, Sep. 2019.

[17] S. Branco, A. G. Ferreira, and J. Cabral, "Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: A survey," *Electronics*, vol. 8, no. 11, p. 1289, Nov. 2019.

[18] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han, and J. Cohn, "MCUNet: Tiny deep learning on IoT devices," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 11711–11722, 2020.

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

[20] A. Pramod, H. S. Naicker, and A. K. Tyagi, "Machine learning and deep learning: Open issues and future research directions for the next 10 years," in *Proc. Comput. Anal. Deep Learn. Med. Care, Princ., Methods, Appl.*, 2021, pp. 463–490.

[21] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," 2020, *arXiv:2007.05558*.

[22] V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the extreme edge: A survey on reformable TinyML," *ACM Comput. Surveys*, vol. 55, no. 13s, pp. 1–30, Dec. 2023.

[23] O. Bringmann, W. Ecker, I. Feldner, A. Frischknecht, C. Gerum, T. Hämäläinen, M. A. Hanif, M. J. Klaiber, D. Mueller-Gritschneder, and P. P. Bernardo, "Automated HW/SW co-design for edge AI: State, challenges and steps ahead," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, 2021, pp. 11–20.

[24] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5149–5169, Sep. 2022.

[25] M. Trimmel, M. Zanfir, R. Hartley, and C. Sminchisescu, "ERA: Enhanced rational activations," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2022, pp. 722–738.

[26] L. Metz, J. Harrison, C. Daniel Freeman, A. Merchant, L. Beyer, J. Bradbury, N. Agrawal, B. Poole, I. Mordatch, A. Roberts, and J. Sohl-Dickstein, "VeLO: Training versatile learned optimizers by scaling up," 2022, *arXiv:2211.09760*.

[27] C. R. Banbury, V. Janapa Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. Patterson, D. Pau, J.-S. Seo, J. Sieracki, U. Thakker, M. Verhelst, and P. Yadav, "Benchmarking TinyML systems: Challenges and direction," 2020, *arXiv:2003.04821*.

[28] R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits Syst. Mag.*, vol. 20, no. 3, pp. 4–18, 3rd Quart., 2020.

[29] Y. Y. Siang, M. R. Ahamd, and M. S. Z. Abidin, "Anomaly detection based on tiny machine learning: A review," *Open Int. J. Informat.*, vol. 9, no. 2, pp. 67–78, 2021.

[30] V. Tsoukas, E. Boumpa, G. Giannakas, and A. Kakarountas, "A review of machine learning and TinyML in healthcare," in *Proc. 25th Pan-Hellenic Conf. Informat.*, Nov. 2021, pp. 69–73.

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

[32] E. Njor, J. Madsen, and X. Fafoutis, "A primer for TinyML predictive maintenance: Input and model optimisation," in *Proc. Artif. Intell. Appl. Innov.* Springer, 2022, pp. 67–78.

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

[34] R. Immonen and T. Hämäläinen, "Tiny machine learning for resource-constrained microcontrollers," *J. Sensors*, vol. 2022, pp. 1–11, Nov. 2022.

[35] N. Schizas, A. Karras, C. Karras, and S. Sioutas, "TinyML for ultra-low power AI and large scale IoT deployments: A systematic review," *Future Internet*, vol. 14, no. 12, p. 363, Dec. 2022.

[36] S. B. Lakshman and N. U. Eisty, "Software engineering approaches for TinyML based IoT embedded vision: A systematic literature review," in *Proc. IEEE/ACM 4th Int. Workshop Softw. Eng. Res. Practices IoT (SERP4IoT)*, May 2022, pp. 33–40.

[37] N. N. Alajlan and D. M. Ibrahim, "TinyML: Enabling of inference deep learning models on ultra-low-power IoT edge devices for AI applications," *Micromachines*, vol. 13, no. 6, p. 851, May 2022.

[38] H. Bamoumen, A. Temouden, N. Benamar, and Y. Chtouki, "How TinyML can be leveraged to solve environmental problems: A survey," in *Proc. Int. Conf. Innov. Intell. Informat., Comput., Technol. (ICT)*, Nov. 2022, pp. 338–343.

[39] W. Su, L. Li, F. Liu, M. He, and X. Liang, "AI on the edge: A comprehensive review," *Artif. Intell. Rev.*, vol. 55, no. 8, pp. 6125–6183, Dec. 2022.

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

[41] S. Prakash, T. Callahan, J. Bushagour, C. Banbury, A. V. Green, P. Warden, T. Ansell, and V. J. Reddi, "CFU playground: Full-stack open-source framework for tiny machine learning (TinyML) acceleration on FPGAS," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2023, pp. 157–167.

[42] S. Kalapothas, M. Galetakis, G. Flamis, F. Plessas, and P. Kitsos, "A survey on RISC-V-based machine learning ecosystem," *Information*, vol. 14, no. 2, p. 64, Jan. 2023.

[43] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on TinyML," *IEEE Access*, vol. 11, pp. 96892–96922, 2023.

[44] J. Yepes-Nuñez, G. Urrutia, M. Romero-Garcia, and S. Alonso-Fernandez, "The PRISMA 2020 statement: An updated guideline for reporting systematic reviews," *Revista Espanola de Cardiologia English ed.*, vol. 74, no. 9, pp. 790–799, 2021.

[45] AIC Computer-Aided Design (ICCAD). (2023). *Tinyml Design Contest*. [Online]. Available: https://tinymlcontest.github.io/TinyML-Design-Contest/

[46] TinyML Foundation. (2023). *Tinyml Challenge 2022: Smart Weather Station*. [Online]. Available: https://www.tinyml.org/event/tinyml-challenge-2022-smart-weather-station-2/

[47] H. Ren, D. Anicic, and T. A. Runkler, "TinyReptile: TinyML with federated meta-learning," 2023, *arXiv:2304.05201*.

[48] L. Heim, A. Biri, Z. Qu, and L. Thiele, "Measuring what really matters: Optimizing neural networks for TinyML," 2021, *arXiv:2104.10645*.

[49] J. Chang, Y. Choi, T. Lee, and J. Cho, "Reducing MAC operation in convolutional neural network with sign prediction," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2018, pp. 177–182.

[50] M. Olyaiy, C. Ng, and M. Lis, "Accelerating DNNs inference with predictive layer fusion," in *Proc. ACM Int. Conf. Supercomputing*, Jun. 2021, pp. 291–303.

[51] W.-C. Lin, Y.-C. Chang, and J.-D. Huang, "An efficient and low-power MLP accelerator architecture supporting structured pruning, sparse activations and asymmetric quantization for edge computing," in *Proc. IEEE 3rd Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2021, pp. 1–5.

[52] Y.-C. Zhou, M. Lei, Y.-L. Zhang, Q. Zhang, and J. Han, "An enhanced data cache with in-cache processing units for convolutional neural network accelerators," in *Proc. IEEE 15th Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Nov. 2020, pp. 1–3.

[53] F. Conti, D. Rossi, A. Pullini, I. Loi, and L. Benini, "Energy-efficient vision on the PULP platform for ultra-low power parallel computing," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2014, pp. 1–6.

[54] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Phil. Trans. Roy. Soc. A: Math., Phys. Eng. Sci.*, vol. 378, no. 2164, Feb. 2020, Art. no. 20190155.

[55] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: A computing library for quantized neural network inference at the edge on RISC-V based parallel ultra low power clusters," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 33–36.

[56] C. Zhou, F. G. Redondo, J. Büchel, I. Boybat, X. T. Comas, S. R. Nandakumar, S. Das, A. Sebastian, M. Le Gallo, and P. N. Whatmough, "ML-HW co-design of noise-robust TinyML models and always-on analog compute-in-memory edge accelerator," *IEEE Micro*, vol. 42, no. 6, pp. 76–87, Nov. 2022.

[57] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016, *arXiv:1607.03250*.

[58] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM J. Emerg. Technol. Comput. Syst. (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.

[59] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*.

[60] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN pruning to the underlying hardware parallelism," in *Proc. ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, 2017, pp. 548–560.

[61] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv:1810.05270*.

[62] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" in *Proc. Mach. Learn. Syst.*, vol. 2, 2020, pp. 129–146.

[63] J. D. De Leon and R. Atienza, "Depth pruning with auxiliary networks for tinyml," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2022, pp. 3963–3967.

[64] S. Vadera and S. Ameen, "Methods for pruning deep neural networks," *IEEE Access*, vol. 10, pp. 63280–63300, 2022.

[65] B. Sun, S. Bayes, A. M. Abotaleb, and M. Hassan, "The case for TinyML in healthcare: CNNs for real-time on-edge blood pressure estimation," in *Proc. 38th ACM/SIGAPP Symp. Appl. Comput.*, Mar. 2023, pp. 629–638.

[66] M. Hashir, N. Khalid, N. Mahmood, M. A. Rehman, M. Asad, M. Q. Mehmood, M. Zubair, and Y. Massoud, "A TinyML based portable, low-cost microwave head imaging system for brain stroke detection," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2023, pp. 1–4.

[67] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave Gaussian quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5918–5926.

[68] B. Jacob, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.

[69] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.

[70] Y. Mishchenko, Y. Goren, M. Sun, C. Beauchene, S. Matsoukas, O. Rybakov, and S. N. P. Vitaladevuni, "Low-bit quantization and quantization-aware training for small-footprint keyword spotting," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 706–711.

[71] M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos, and T. Blankevoort, "Up or down? Adaptive rounding for post-training quantization," in *Proc. Int. Conf. Machine. Learn. (ICML)*, 2020, pp. 7197–7206.

[72] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9847–9856.

[73] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Boca Raton, FL, USA: Chapman & Hall/CRC, 2022, pp. 291–326.

[74] M. Nagel, M. Fournarakis, Y. Bondarenko, and T. Blankevoort, "Overcoming oscillations in quantization-aware training," 2022, *arXiv:2203.11086*.

[75] S. Zhuo, H. Chen, R. Kinattinkara Ramakrishnan, T. Chen, C. Feng, Y. Lin, P. Zhang, and L. Shen, "An empirical study of low precision quantization for TinyML," 2022, *arXiv:2203.05492*.

[76] J. Moosmann, M. Giordano, C. Vogt, and M. Magno, "TinyissimoYOLO: A quantized, low-memory footprint, TinyML object detection network for low power microcontrollers," 2023, *arXiv:2306.00001*.

[77] Q. Lu and B. Murmann, "Enhancing the energy efficiency and robustness of TinyML computer vision using coarsely-quantized log-gradient input images," *ACM Trans. Embedded Comput. Syst.*, Apr. 2023.

[78] N. N. Alajlan and D. M. Ibrahim, "DDD TinyML: A TinyML-based driver drowsiness detection model using deep learning," *Sensors*, vol. 23, no. 12, p. 5696, Jun. 2023.

[79] S. Yun, J. Park, K. Lee, and J. Shin, "Regularizing class-wise predictions via self-knowledge distillation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 13876–13885.

[80] H. Zhao, X. Sun, J. Dong, C. Chen, and Z. Dong, "Highlight every step: Knowledge distillation via collaborative teaching," *IEEE Trans. Cybern.*, vol. 52, no. 4, pp. 2070–2081, Apr. 2022.

[81] L. Zhang and K. Ma, "Improve object detection with feature-based knowledge distillation: Towards accurate and efficient detectors," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–12.

[82] N. Körber, A. Siebert, S. Hauke, and D. Mueller-Gritschneder, "Tiny generative image compression for bandwidth-constrained sensor applications," in *Proc. 20th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2021, pp. 564–569.

[83] A. Ukil, I. Sahu, A. Majumdar, S. C. Racha, G. Kulkarni, A. D. Choudhury, S. Khandelwal, A. Ghose, and A. Pal, "Resource constrained CVD classification using single lead ECG on wearable and implantable devices," in *Proc. 43rd Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Nov. 2021, pp. 886–889.

[84] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, Jun. 2021.

[85] H. Zhang, Z. Hu, W. Qin, M. Xu, and M. Wang, "Adversarial co-distillation learning for image recognition," *Pattern Recognit.*, vol. 111, Mar. 2021, Art. no. 107659.

[86] H. Cheng, L. Yang, and Z. Liu, "Relation-based knowledge distillation for anomaly detection," in *Proc. Chin. Conf. Pattern Recognit. Comput. Vis. (PRCV)*. Springer, 2021, pp. 105–116.

[87] X. Dai, "General instance distillation for object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 7842–7851.

[88] H. A. Zein, M. Aoude, and Y. Harkous, "Implementation and optimization of neural networks for tiny hardware devices," in *Proc. Int. Conf. Smart Syst. Power Manage. (IC2SPM)*, Nov. 2022, pp. 191–196.

[89] A. Brutti, F. Paissan, A. Ancilotto, and E. Farella, "Optimizing PhiNet architectures for the detection of urban sounds on low-end devices," in *Proc. 30th Eur. Signal Process. Conf. (EUSIPCO)*, Aug. 2022, pp. 1121–1125.

[90] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 24, 2011, pp. 1–12.

[91] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 2, pp. 1–25, 2012.

[92] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on Bayesian optimization," *J. Electron. Sci. Technol.*, vol. 17, pp. 26–40, Mar. 2019.

[93] S. S. Sandha, M. Aggarwal, S. S. Saha, and M. Srivastava, "Enabling hyperparameter tuning of machine learning classifiers in production," in *Proc. IEEE 3rd Int. Conf. Cognit. Mach. Intell. (CogMI)*, Dec. 2021, pp. 262–271.

[94] C. Tang, K. Ouyang, Z. Wang, Y. Zhu, W. Ji, Y. Wang, and W. Zhu, "Mixed-precision neural network quantization via learned layer-wise importance," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2022, pp. 259–275.

[95] J. Youn, J. Song, H.-S. Kim, and S. Bahk, "Bitwidth-adaptive quantization-aware neural network training: A meta-learning approach," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2022, pp. 208–224.

[96] H. R. Mendis, C.-K. Kang, and P.-C. Hsiu, "Intermittent-aware neural architecture search," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5s, pp. 1–27, Oct. 2021.

[97] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers," in *Proc. Mach. Learn. Syst.*, vol. 3, 2021, pp. 517–532.

[98] E. Liberis, Ł. Dudziak, and N. D. Lane, "$\mu$nas: Constrained neural architecture search for microcontrollers," in *Proc. 1st Workshop Mach. Learn. Syst.*, 2021, pp. 70–79.

[99] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Survey*, vol. 54, no. 4, pp. 1–34, 2021.

[100] D. Baymurzina, E. Golikov, and M. Burtsev, "A review of neural architecture search," *Neurocomputing*, vol. 474, pp. 82–93, Feb. 2022.

[101] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 2, pp. 550–570, Feb. 2023.

[102] E. Njor, J. Madsen, and X. Fafoutis, "Data aware neural architecture search," 2023, *arXiv:2304.01821*.

[103] D. P. Pau, P. K. Ambrose, and F. M. Aymone, "A quantitative review of automated neural search and on-device learning for tiny devices," *Chips*, vol. 2, no. 2, pp. 130–141, May 2023.

[104] A. M. Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "A hardware-aware neural architecture search algorithm targeting low-end microcontrollers," in *Proc. 18th Conf. Ph.D Res. Microelectron. Electron. (PRIME)*, Jun. 2023, pp. 281–284.

[105] A. Molina, P. Schramowski, and K. Kersting, "Padé activation units: End-to-end learning of flexible activation functions in deep networks," 2019, *arXiv:1907.06732*.

[106] A. Apicella, F. Donnarumma, F. Isgro, and R. Prevete, "A survey on modern trainable activation functions," *Neural Netw.*, vol. 138, pp. 14–32, Jun. 2021.

[107] 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," 2017, *arXiv:1704.04861*.

[108] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.

[109] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[110] J. Luo, J. Wang, N. Cheng, E. Xiao, X. Zhang, and J. Xiao, "Tiny-sepformer: A tiny time-domain transformer network for speech separation," 2022, *arXiv:2206.13689*.

[111] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–12.

[112] L. Huang, W. Wang, J. Chen, and X.-Y. Wei, "Attention on attention for image captioning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 4634–4643.

[113] A. Galassi, M. Lippi, and P. Torroni, "Attention in natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 10, pp. 4291–4308, May 2020.

[114] A. Wong, M. Famouri, and M. Javad Shafiee, "AttendNets: Tiny deep image recognition neural networks for the edge via visual attention condensers," 2020, *arXiv:2009.14385*.

[115] G. Brauwers and F. Frasincar, "A general survey on attention mechanisms in deep learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 3279–3298, Apr. 2023.

[116] A. Burrello, M. Scherer, M. Zanghieri, F. Conti, and L. Benini, "A microcontroller is all you need: Enabling transformer execution on low-power IoT endnodes," in *Proc. IEEE Int. Conf. Omni-Layer Intell. Syst. (COINS)*, Aug. 2021, pp. 1–6.

[117] S. Mehta and M. Rastegari, "MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer," 2021, *arXiv:2110.02178*.

[118] S. Abbasi Koohpayegani, A. Tejankar, and H. Pirsiavash, "Compress: Self-supervised learning by compressing representations," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 12980–12992.

[119] M. de Prado, M. Rusci, A. Capotondi, R. Donze, L. Benini, and N. Pazos, "Robustifying the deployment of TinyML models for autonomous mini-vehicles," *Sensors*, vol. 21, no. 4, p. 1339, Feb. 2021.

[120] D. Hussein and G. Bhat, "SensorGAN: A novel data recovery approach for wearable human activity recognition," *ACM Trans. Embedded Comput. Syst.*, Jul. 2023.

[121] N. Yamin and G. Bhat, "Uncertainty-aware energy harvest prediction and management for IoT devices," *ACM Trans. Design Autom. Electron. Syst.*, vol. 28, no. 5, pp. 1–33, Sep. 2023.

[122] S. Goyal, A. Raghunathan, M. Jain, H. V. Simhadri, and P. Jain, "DROCC: Deep robust one-class classification," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3711–3721.

[123] S. Abbasi, M. Famouri, M. J. Shafiee, and A. Wong, "OutlierNets: Highly compact deep autoencoder network architectures for on-device acoustic anomaly detection," *Sensors*, vol. 21, no. 14, p. 4805, Jul. 2021.

[124] H. Kayan, Y. Majib, W. Alsafery, M. Barhamgi, and C. Perera, "AnoML-IoT: An end to end re-configurable multi-protocol anomaly detection pipeline for Internet of Things," *Internet Things*, vol. 16, Dec. 2021, Art. no. 100437.

[125] M. Lord and A. Kaplan, "Mechanical anomaly detection on an embedded microcontroller," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2021, pp. 562–568.

[126] P. Andrade, I. Silva, G. Signoretti, M. Silva, J. Dias, L. Marques, and D. G. Costa, "An unsupervised TinyML approach applied for pavement anomalies detection under the Internet of Intelligent vehicles," in *Proc. IEEE Int. Workshop Metrol. Ind. 4.0 IoT (MetroInd4. 0IoT)*, 2021, pp. 642–647.

[127] P. Andrade, I. Silva, M. Silva, T. Flores, J. Cassiano, and D. G. Costa, "A TinyML soft-sensor approach for low-cost detection and monitoring of vehicular emissions," *Sensors*, vol. 22, no. 10, p. 3838, May 2022.

[128] X. Liu, "Self-supervised learning: Generative or contrastive," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 857–876, Jun. 2021.

[129] M. Abououf, R. Mizouni, S. Singh, H. Otrok, and E. Damiani, "Self-supervised online and lightweight anomaly and event detection for IoT devices," *IEEE Internet Things J.*, vol. 9, no. 24, pp. 25285–25299, Dec. 2022.

[130] V. Rani, S. T. Nabi, M. Kumar, A. Mittal, and K. Kumar, "Self-supervised learning: A succinct review," *Arch. Comput. Methods Eng.*, vol. 30, no. 4, pp. 2761–2775, May 2023.

[131] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.

[132] F. Svoboda, D. Nunes, M. Alizadeh, R. Daries, R. Luo, A. Mathur, S. Bhattacharya, J. S. Silva, and N. D. Lane, "Resource efficient deep reinforcement learning for acutely constrained tinyml devices," in *Proc. Res. Symp. Tiny Mach. Learn.*, 2020.

[133] T. Szydlo, P. P. Jayaraman, Y. Li, G. Morgan, and R. Ranjan, "TinyRL: Towards reinforcement learning on tiny embedded devices," in *Proc. 31st ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2022, pp. 4985–4988.

[134] X. Kong, "Deep reinforcement learning-based energy-efficient edge computing for Internet of Vehicles," *IEEE Trans. Ind. Informat.*, vol. 18, no. 9, pp. 6308–6316, Sep. 2022.

[135] D. Pau, S. Colella, and C. Marchisio, "End to end optimized tiny learning for repositionable walls in maze topologies," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2023, pp. 1–7.

[136] T. Barbariol and G. A. Susto, "TiWS-iForest: Isolation forest in weakly supervised and tiny ML scenarios," *Inf. Sci.*, vol. 610, pp. 126–143, Sep. 2022.

[137] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "An adaptable and unsupervised TinyML anomaly detection system for extreme industrial environments," *Sensors*, vol. 23, no. 4, p. 2344, Feb. 2023.

[138] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[139] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine, "Model-based meta-reinforcement learning for flight with suspended payloads," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 1471–1478, Apr. 2021.

[140] H. Cho, Y. Cho, J. Yu, and J. Kim, "Camera distortion-aware 3D human pose estimation in video with optimization-based meta-learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 11149–11158.

[141] D. Gao, X. He, Z. Zhou, Y. Tong, and L. Thiele, "Pruning meta-trained networks for on-device adaptation," in *Proc. ACM Int. Conf. Inf. Knowl. Manag.*, 2021, pp. 514–523.

[142] D. Gao, Y. Xie, Z. Zhou, Z. Wang, Y. Li, and B. Ding, "Finding meta winning ticket to train your MAML," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2022, pp. 411–420.

[143] Y. Liu, S. Duan, X. Xu, and S. Ren, "MetaLDC: Meta learning of low-dimensional computing classifiers for fast on-device adaption," 2023, *arXiv:2302.12347*.

[144] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, and A. Grabska-Barwinska, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017.

[145] H. Cai, C. Gan, L. Zhu, and S. Han, "TinyTL: Reduce memory, not parameters for efficient on-device learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 11285–11297.

[146] H. Ren, D. Anicic, and T. A. Runkler, "TinyOL: TinyML with online-learning on microcontrollers," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2021, pp. 1–8.

[147] L. Ravaglia, M. Rusci, D. Nadalini, A. Capotondi, F. Conti, and L. Benini, "A TinyML platform for on-device continual learning with quantized latent replays," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 4, pp. 789–802, Dec. 2021.

[148] B. Sudharsan, P. Yadav, J. G. Breslin, and M. I. Ali, "Train++: An incremental ML model training algorithm to create self-learning IoT devices," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, 2021, pp. 97–106.

[149] A. Avi, A. Albanese, and D. Brunelli, "Incremental online learning algorithms comparison for gesture and visual smart sensors," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2022, pp. 1–8.

[150] M. Pavan, E. Ostrovan, A. Caltabiano, and M. Roveri, "TyBox: An automatic design and code-generation toolbox for TinyML incremental on-device learning," *ACM Trans. Embedded Comput. Syst.*, Jun. 2023.

[151] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[152] N. Suda and D. Loh, *Machine Learning on Arm Cortex-m Microcontrollers*. Cambridge, U.K.: Arm Ltd, 2019.

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

[154] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012.

[155] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA tensor core programmability, performance & precision," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, 2018, pp. 522–531.

[156] Y. Emma Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU platforms for deep learning," 2019, *arXiv:1907.10701*.

[157] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[158] N. Sulaiman, M. H. Marhaban, and M. N. Hamidon, "Design and implementation of FPGA-based systems—A review," *Austral. J. Basic Appl. Sci.*, vol. 3, no. 4, pp. 3575–3596, 2009.

[159] T. Wang, C. Wang, X. Zhou, and H. Chen, "A survey of FPGA based deep learning accelerators: Challenges and opportunities," 2018, *arXiv:1901.04988*.

[160] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, and A. Borchers, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 1–12.

[161] K. Seshadri, B. Akin, J. Laudon, R. Narayanaswami, and A. Yazdanbakhsh, "An evaluation of edge TPU accelerators for convolutional neural networks," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2022, pp. 79–91.

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

[163] uTensor. (2023). *Utensor: Tinyml Ai Inference Library*. [Online]. Available: https://utensor.github.io/website/

[164] V. Janapa Reddi, A. Elium, S. Hymel, D. Tischler, D. Situnayake, C. Ward, L. Moreau, J. Plunkett, M. Kelcey, and M. Baaijens, "Edge impulse: An MLOPS platform for Tiny machine learning," in *Proc. Mach. Learn. Syst.*, vol. 5, 2023, pp. 1–15.

[165] STMicroelectronics. (2023). *X-Cube-AI: AI Expansion Pack for STM32CUBEMX*. [Online]. Available: https://www.st.com/en/embedded-software/x-cube-ai.html

[166] Microsoft. (2023). *Embedded Learning Library: An Open Source Library for Embedded Ai and Machine Learning From Microsoft*. [Online]. Available: https://microsoft.github.io/ELL/

[167] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, and L. Ceze, "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2018, pp. 578–594.

[168] S. Jaiswal, R. K. K. Goli, A. Kumar, V. Seshadri, and R. Sharma, "MinUn: Accurate ML inference on microcontrollers," in *Proc. 24th ACM SIGPLAN/SIGBED Int. Conf. Lang., Compil., Tools Embedded Syst.*, Jun. 2023, pp. 26–39.

[169] P. Mattson, C. Cheng, G. Diamos, C. Coleman, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, and V. Bittorf, "MLPerf training benchmark," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 336–349, Mar. 2020.
[170] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, and W. Chou, "MLPerf inference benchmark," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2020, pp. 446–459.
[171] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, and D. Pau, "MLPerf tiny benchmark," 2021, *arXiv:2106.07597*.
[172] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, Jun. 2009, pp. 248–255.

**LUIGI CAPOGROSSO** (Student Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science and computer engineering for robotics and smart industry from the University of Verona, in 2019 and 2021, respectively, where he is currently pursuing the Ph.D. degree with the National Program in Artificial Intelligence, in collaboration with Politecnico di Torino in the IntelliGO labs under the supervision of Prof. Marco Cristani and Prof. Franco Fummi. His research interests efficient deep learning and representation learning.

**FEDERICO CUNICO** (Graduate Student Member, IEEE) received the master's degree in computer science and engineering from the University of Verona, Verona, Italy, in 2019, where he is currently pursuing the Ph.D. degree in computer science, under the supervision of Prof. Marco Cristani. His main research interests include deep learning and computer vision for industrial scene analysis, emphasizing human-centered tasks, such as attention estimation, human pose and motion forecasting, and human pose estimation.

**DONG SEON CHENG** received the Laurea and Ph.D. degrees in computer science from the University of Verona, Verona, Italy, in 2005 and 2008, respectively, with a focus on computer vision and pattern recognition. From 2012 to 2017, he was an Assistant Professor with the Department of Computer Science and Engineering, Hankuk University of Foreign Studies, South Korea, teaching undergraduate and graduate courses. From 2019 to 2022, he was with SETECNA EPC S.rl., an electronics company in the HVAC industry. Since 2023, he has been a Research Associate with the University of Verona. His research interest includes machine learning. He has published several conferences and journal articles in this field.

**FRANCO FUMMI** (Member, IEEE) received the Laurea degree in electronic engineering and the Ph.D. degree in electronic and communication engineering from Politecnico di Milano, Milan, Italy, in 1990 and 1994, respectively. In 1993, he was a Research Assistant with the Department of Computer Science, University of Victoria. In 1996, he was an Assistant Professor in computer science with the Dipartimento di Elettronica e Informazione, Polytechnic of Milan, where he remained, until October 1998. In July 1998, he was an Associate Professor in computer architecture with the Department of Computer Science, University of Verona. Since March 2001, he has been a Full Professor in computer architecture with the University of Verona, where he is leading the Cyber-Physical and IoT Systems Design (CISD) Group, currently composed of more than 20 people, and working on hardware description languages and electronic design automation methodologies for modeling, verification, testing, and optimization of cyber-physical systems. Prior to this, he was with the Department of Computer Science and then with the Department of Engineering for Innovation Medicine, University of Verona. Since 2018, he has been the Project Manager of the Industrial Computer Engineering (ICE) Laboratory, University of Verona: a facility serving as a technological demonstrator and research laboratory, functional to rethink industrial processes such as additive and subtractive manufacturing, quality control, assembly, and parts storage. He is also the Co-Founder of two spin-off companies, such as EDALab, which focuses on networked embedded systems design, and the automation control software company FACTORYAL.

**MARCO CRISTANI** (Member, IEEE) is currently a Full Professor with the Department of Engineering for Innovation Medicine, University of Verona, an Associate Member with the National Research Council (CNR), and an External Collaborator with the Italian Institute of Technology (IIT). He has organized 11 international workshops and cofounded two spin-off companies, such as Humatics, which deals with e-commerce for fashion, and QUALYCO, which deals with quality control through artificial intelligence in the industrial field. He is or has been a principal investigator of several national and international projects, including PRIN and H2020 projects. His main research interest includes statistical pattern recognition and computer vision, mainly in deep learning and generative modeling, applying to social signal processing and fashion modeling. On these topics, he has published more than 200 articles. He is an IAPR fellow.