A Review on the emerging technology of TinyML

VASILEIOS TSOUKAS*, Computer Science and Biomedical Informatics, University of Thessaly School of Science, Lamia, Greece and Intelligent Systems Laboratory, University of Thessaly School of Science, Lamia, Greece
ANARGYROS GKOGKIDIS*, Computer Science and Biomedical Informatics, University of Thessaly School of Science, Lamia, Greece and Intelligent Systems Laboratory, University of Thessaly School of Science, Lamia, Greece
ELENI BOUMPA, Computer Science and Biomedical Informatics, University of Thessaly School of Science, Lamia, Greece and Intelligent Systems Laboratory, University of Thessaly School of Science, Lamia, Greece
ATHANASIOS KAKAROUNTAS, Computer Science and Biomedical Informatics, University of Thessaly School of Science, Lamia, Greece and Intelligent Systems Laboratory, University of Thessaly School of Science, Lamia, Greece

Tiny Machine Learning (TinyML) is an emerging technology proposed by the scientific community for developing autonomous and secure devices that can gather, process, and provide results without transferring data to external entities. The technology aims to democratize AI by making it available to more sectors and contribute to the digital revolution of intelligent devices. In this work, a classification of the most common optimization techniques for Neural Network compression is conducted. Additionally, a review of the development boards and TinyML software is presented. Furthermore, the work provides educational resources, a classification of the technology applications, and future directions and concludes with the challenges and considerations.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → *Neural networks*; *Embedded software*; *Embedded hardware*; • **Software and its engineering** → *Embedded software*; • **Computing methodologies** → **Machine learning approaches**.

Additional Key Words and Phrases: TinyML, Machine Learning, Neural Networks, Edge AI, resource-constrained intelligence, microcontrollers, constrained hardware, optimization

*Both authors contributed equally to this research.

Authors' Contact Information: Vasileios Tsoukas, Computer Science and Biomedical Informatics, University of Thessaly School of Science, Lamia, Central Greece, Greece and Intelligent Systems Laboratory, University of Thessaly School of Science, Lamia, Central Greece, Greece; e-mail: vtsoukas@uth.gr; Anargyros Gkogkidis, Computer Science and Biomedical Informatics, University of Thessaly School of Science, Lamia, Central Greece, Greece and Intelligent Systems Laboratory, University of Thessaly School of Science, Lamia, Central Greece, Greece; e-mail: agkogkidis@uth.gr; Eleni Boumpa, Computer Science and Biomedical Informatics, University of Thessaly School of Science, Lamia, Central Greece, Greece and Intelligent Systems Laboratory, University of Thessaly School of Science, Lamia, Central Greece, Greece; e-mail: eboumpa@uth.gr; Athanasios Kakarountas, Computer Science and Biomedical Informatics, University of Thessaly School of Science, Lamia, Central Greece, Greece and Intelligent Systems Laboratory, University of Thessaly School of Science, Lamia, Central Greece, Greece; e-mail: kakarountas@uth.gr.

ACM 1557-7341/2024/5-ART
https://doi.org/10.1145/3661820

1 INTRODUCTION

Machine Learning (ML) is a rapidly growing study topic in academia and industry. When combined with the fields of the Internet of Things (IoT), Data Science, and the fourth industrial revolution (Industry 4.0), ML research entails a massive development of solutions with a variety of applications in diverse and multidisciplinary fields, including medical contexts, pattern recognition, finance, and environmental science [135]. The availability of large-scale data extraction paired with advancements in hardware technology has resulted in the development of a well-known and commonly used machine learning subset, Deep Learning (DL). The pattern of DL is similar to that of the brain's Neural Networks (NNs), and two of the most significant elements regarding NNs are the automated feature extraction and analysis of unstructured data. The most widely used DL network types are Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory Networks (LSTMs) [190].

The high computational complexity of DL networks has led researchers to concentrate their efforts on compression techniques and optimization methods capable of increasing the efficiency of the NNs while maintaining the same levels of accuracy and performance. The aforementioned solutions are software-based. Contrastingly, other attempts are focused on the hardware design principles, to accelerate the overall process, utilizing hardware components that allow complex matrix functions [147, 186]. For example, Application Specific Integrated Circuits (ASICs) are designed with specific architecture to achieve the acceleration of ML applications workload, providing support for fixed-point calculations, and floating-point (FP) calculations. Also, Field-Programmable Gate Arrays (FPGAs), are reprogrammable ASICs, that provide high performance concerning energy consumption compared to Graphics Processing Units (GPUs), and low latency compared to models executed on Central Processing Units (CPUs) [104]. The most notable challenges regarding DL implementations are the high cost of the training models and the high computational requirements of the hardware required to process data and extract results.

The wide usage of both ML and DL methods in various fields allows for analyzing information from a large amount of data and extracting valuable insights. In a typical IoT ecosystem, data is gathered, transferred, processed, and stored with the goal of monitoring values and preventing or forecasting emergencies. Data is acquired from interconnected devices, such as sensors and actuators, and then sent to the cloud, where it is processed and stored. IoT gateways are used to connect the sensors' network to the cloud network. The cloud serves as the central point of the IoT ecosystem, processing huge amounts of data and delivering intelligent ML or DL-based applications [58, 188]. The majority of IoT applications require fast response times, wireless and high-speed data transmission, high network bandwidth, low latency, and data security.

While the cloud offers necessary processing power and decision-making mechanisms, transmitting data to it carries risks concerning data privacy and security. Notably, data on the cloud may not always be secure, exposing sensitive information to potential threats like man-in-the-middle attacks, replay attacks, eavesdropping, and rogue access points.[21, 91, 137, 164, 184].

In contrast, edge computing offers a safer alternative by providing cloud-like services directly at the network's edge. This method, named for its decentralization of computational power, enables faster data processing, greater bandwidth, and data sovereignty [108, 230]. Furthermore, over the last decades, researchers concentrated on embedding ML models on edge devices and constrained hardware [176]. The Microcontroller Unit (MCU) is the hardware utilized for achieving the aforementioned operation [45]. The high level of interest in MCUs is attributed to their specifications, which include essential performance [134], low power consumption, and a tiny form factor[58].

New emerging trends utilizing ML to provide tailored and personalized results in IoT devices are federated learning and Tiny Machine Learning (TinyML). In federated learning, participants cooperatively train ML models by exchanging model parameters, allowing for personalized training for each participant while safeguarding their sensitive data. The TinyML approach is a hardware-software hybrid focused on embedding ML models and NNs

on constrained hardware capable of processing them locally [103]. The most significant advantage of the TinyML approach is the real-time analysis of the collected data while overcoming latency and bandwidth limitations [23]. Additionally, the IoT devices that embed TinyML on their MCUs require less access to the cloud services, resulting in cost and power reduction, and incremental data security and privacy [58]. All the benefits mentioned above, have led to the development of several TinyML-based applications in various fields, such as healthcare, automotive, agriculture, security, and the industry. Furthermore, TinyML's popularity and quick expansion led to the introduction of several TinyML-compatible development boards, frameworks, libraries, and other toolkits.

Deep Neural Networks (DNNs) are constantly growing in size and trying to achieve the best possible performance to handle and process complicated issues in fields such as robotics, Natural Language Processing (NLP), security, and Computer Vision (CV). Two of the most common sectors that utilize DNNs are CV and NLP. Transformer-based architectures for the aforementioned sectors [54, 118, 126, 158, 177, 212, 229] typically have many layers where each layer has millions of parameters [30, 189]. The computational resources required for training, processing, and storing CNNs of that scale and complexity necessitate using powerful scientific workstations with high-end GPUs and numerous CPUs. Additionally, with the advances of IoT, mobile, and wearable devices, a new need arises that motivates researchers to compress and optimize the networks stated above to be compatible with constrained hardware with limited resources. Moreover, since networks and models keep expanding to offer better performance, even some high-end machines do not have the required power to handle them, resulting in many issues regarding the ML professionals and how the research community could provide solutions by offering better optimization techniques. This review doesn't cover DNNs implementation for constrained hardware due to the extension of the issue and page limitations of the journal.

The technology of TinyML is in its early stages of development, hence, it lacks a number of capabilities and has several limitations. It is difficult to provide a common framework for model training and inference compatible with every MCU. Due to the heterogeneity of the devices, the models that are built expressly for one MCU architecture may not necessarily run on another, even if they share the same hardware and specifications. Finally, researchers must comprehend the relationship between power consumption and processing speed, as well as their impact on algorithm accuracy [194].

The contribution of this study is to provide a review of the TinyML approach in IoT devices. The remainder of this work is organized as follows. Section 2 provides an overall discussion of the TinyML technology, showcasing an overview of compression ML techniques. Section 3 reveals TinyML's impact over the years and provides useful educational resources. Section 4 consists of subsection 4.1 that presents a review of TinyML software, and subsection 4.2 that presents a review of TinyML hardware, while Section 5 provides a categorized review of TinyML applications and the future directions of the technology under consideration. Finally, Section 6 discusses the challenges of the technology of TinyML, while Section 7 concludes this study.

2 THE TECHNOLOGY OF TINYML

TinyML is a hybrid software-hardware technology with high scientific and industrial interest due to its potential for creating small, autonomous, and energy efficient smart devices [70, 72]. TinyML can be identified as a cluster of technologies working harmoniously to provide a necessary result. It is probably the most significant paradigm of technologies requiring a hardware/software co-design flow to unleash the sector's full potential. Inference on edge is an indispensable combination of the frameworks [49], model reduction techniques [220], libraries utilization [81, 116], architecture searches [70], and designing and deployment models to the appropriate hardware [24]. It can be easily observed that there is a high amount of complexity due to the requirements of all the aforementioned technologies. Innovation and continuous progress, especially in the hardware field, are of great importance, with the latest advancement showing encouraging and promising results [63]. The first applications utilizing the technology in consideration have already been introduced [176], and this is a positive

vision of how TinyML could play a significant role in future applications. A brief description of the main benefits of the TinyML technology follows:

(1) Information security and latency: Nowadays, handheld or wearable devices require a secure connection to a cloud service in order to process data and provide results [188]. The first two main issues that come to mind are latency and bandwidth constraints [2, 133, 156]. Additionally, due to their small form factor, power, and energy constraints, most wearable devices are built without a significant focus on security [184] or are in need of real-time responses [130]. Devices such as medical or activity tracking may contain sensitive and private information and may be vulnerable to malicious users and attacks [136, 143, 191, 221]. Cloud providers offer the processing power required for result and decision extraction in a reasonable time frame. That transmission lurks many dangers since private and sensitive information is prone to malicious users and attacks such as Man In The Middle (MITM) attacks, replay attacks, eavesdropping [91, 137, 184] and rogue access point attacks [21, 164]. Most of the time, the transmitted data is not encrypted, and the devices in question have weak or no wireless security features. Furthermore, connecting a user's personal device to business networks can be harmful. Due to various device weaknesses, the device may operate as a starting point for a network backdoor when obtaining corporate data. Additionally, in sectors such as automotive or healthcare, where seconds are crucial, high latency or delays in response time are unacceptable. A TinyML device does not require any data transmission, and a blend of optimized software and hardware could be in place to offer almost instant results.
(2) Energy efficiency and overall cost: A TinyML-based device can be a small form factor, a low-cost, and low-powered autonomous device able to collect data from its sensors, perform the processing, and extract relatively fast results [49]. Moreover, Deep Reinforcement Learning (DRL) is another area of great scientific interest that has started to be explored in ways to be implemented in resource-constrained systems, as shown in recent literature [56, 117, 173, 198, 213]. The complexity of ML tasks requires sufficient process power, translating into power-demanding and expensive equipment that is most commonly met in high-end GPUs [112]. TinyML might change how we visualize those tasks by offering almost the same results by inferring models and networks into MCUs. An MCU is a low-powered device with typically low production cost that can extract decisions and offer machine intelligence with the cost of a single battery. All this is achieved due to the appropriate co-design of software and hardware and the aggressive optimization techniques implemented in the algorithms.

2.1 Optimization Methods

NNs tend to have several parameters with significant redundancy regarding the models, ultimately leading to more computation power and memory usage than required [52]. As stated above, TinyML is a paradigm that allows ML models to fit into constrained hardware without compromising their energy efficiency [55]. To achieve model inference in devices with limited resources, specifically MCUs, the models under consideration must go under optimization and compressing techniques. Optimizing algorithms and ML models is a challenging issue. As described previously, it is not just a software or hardware problem, but hardware and software co-design is a prerequisite to obtain the targeted result [58]. Over the last decades, many methods, frameworks, and techniques have been proposed for compression. O'Neill's work [147] provides an analytical review of those methods. Furthermore, recent research also reports the first attempts to optimize deep RL designed for resource-constrained systems [198]. A brief description of the most commonly used techniques follows, aiming to highlight the preferred ones in TinyML.

2.2 Quantization

A network trained on high-end GPUs has values stored in 32-bit FP single precision. For faster inference and lower computational needs, researchers have focused on training lower-precision networks with one or 2-bit representations able to run on other types of hardware such as FPGAs and ASICs. The process of representing network values with fewer bits is known as quantization. One necessity of the aforementioned method is to retain accuracy or have the lowest possible trade-off. A typical quantization technique lowers the values from FP32 to 8-bit representations [53, 86]. Gupta et al. [85] analyzed that stochastic rounding methods can maintain accuracy when an FP32 model is quantized to FP16. Cambier et al. [33] proposed a shifted and squeezed 8-bit (S2FP-8) to avoid stochastic rounding, while Mellempudi et al. [138] showcased a different approach where the first and last layer is in a different representation than the typical FP32. Park et al. [157] proposed 3-bit activations where weights are also quantized to 4-bit and 16-bit scaling factors for approximately 1% of the network resulting in only 1% of accuracy loss. Migacz proposed different approaches [140], who used relative entropy to measure information loss, and Banner et al. [26], who used noise and clipping distortion. Merolla et al. [139] tried to understand how different distortions affect the networks. Their results revealed that NNs are robust to distortions but have more extensive convergence times. Works [54, 111] suggested that post-training quantization on small networks can hit accuracy when utilizing 8-bit formats or lower. In an attempt to weather the aforementioned issue, Zhou et al. [236] proposed the quantization of gradients to 6-bit and stochastically propagated using estimators. By minimizing the loss regarding binarized weights, Hou et al. [98] proposed a Newton algorithm combined with Hessian approximation. Explicit Loss-Aware Quantization (ELQ) is another method of quantization proposed by Zhou et al. [235], which focuses on minimizing the loss for very low precision. Similarly, Park et al. [157] achieved reduced precision by intensively narrowing the range weight values. Stock et al. [197], to overcome quantization drift, used iterative quantization starting with low layers and performed gradient updates to the rest. On the contrary, Jacob et al.[54] used an estimator to backpropagate through activations and weights during the training process instead of exploiting the aforementioned iterative approach. Finally, Fan et al. [69] suggested that utilizing the previous two approaches is not advisable when operating with ternary or binary precision representations. They proposed a simulation of quantization noise, which is random for a subset of the network, and then the performance of backward weights passes on to the rest.

Quantization is one of the most often used optimization strategies for TinyML compression on MCUs. As described previously, the former attempts to map 64-bit or 32-bit weights to a smaller bit width to store the required models in MCUs [58], while there is no focus or strategies in the initial architecture and instead relies on techniques such as transfer learning after the final model is ready [3]. In work [168], the Quantised Latent Replay-based Continual Learning method is proposed, which relies on low bandwidth quantization that can reduce the memory requirements of the layer but also increase the overall speed of the network. This work is one of the first attempts at creating a hardware and software platform for TinyML continual learning.

2.3 Pruning

Network pruning is one of the oldest and most used techniques for optimization. It is based on the human brain where irrelevant and unimportant past experiences are removed to make room for newer [219]. Network pruning removes synapses and neurons that fall under a certain weight threshold [90] to improve performance and reduce the computational needs of a network without sacrificing a significant amount of accuracy. Han and Qiao [92], and Narasimha et al. [146] suggested a different perspective by combining different techniques, such as the addition of new neurons and a predefined percentage of weights that need to be pruned instead of the standard version of setting a threshold. Magnitude-based Pruning (MBP) is used due to its high performance while it keeps its simplicity for ML and other tasks [185], and even tends to outperform layer-wise MBP [90, 107, 120, 171]. In the works [94, 119, 171], the authors measure the importance of the pruned units and try to remove only those

with minor importance that ultimately will lead to the least loss. Mozer and Smolensky proposed similar works [144] where with their method, skeletonization, proceeded to remove the least relevant units during training, Karnin [107], who measured the sensitivity of the loss function, and Engelbrecht [66], who suggested to check if the variance of sensitivity is not significantly different from zero before proceeding into the removal of weights. Additionally, LeCun et al. [119] proposed that weight importance can be estimated utilizing the Taylor series.

Based on this technique, Molchanov et al. [142] used a Taylor expansion to prune the weights with the lowest change regarding the cost function. Theis et al. [205] extended Molchanov's work by providing cost estimates for network pruning. Mallya and Lazebnik [132] suggested that a dynamic mask can learn to adapt a dense network to a different sparse network. A different approach is structured sparsity learning, which can be found in the work [224]. Louizos et al. [129] suggested that Bayesian methods can also be utilized for structured pruning. Dai et al. [47] proposed an alternative method known as pruning via variational information bottleneck, where the authors aimed to remove mutual information between layers.

Another approach was introduced by Lin et al. [124] by applying a soft mask to minimize the mean squared error. This work is an extension of a previous work [123], in which the authors achieved optimization using binary masks and hard thresholding. Genetic Algorithms (GAs) are another approach that can be utilized for the pruning method by keeping the best performance parameters they generate and mixing them until they achieve the desired result. Several researchers attempted the usage of GAs in order to prune a network, and some examples can be studied in the following works [34, 100, 225].

Noy et al. [150] tried to reduce the required time for searching neural architecture by implementing pruning via simulated pruning. Their work is based on the DARTS approach suggested in [125]. Particle filtering for pruning is another method where sequential Monte Carlo estimation is utilized to identify the crucial weights [9]. Particle swarm optimization for the pruning method was also proposed in [210]. Furthermore, AutoML [95] was proposed to improve the compression performance and the overall efficiency in a more automated pruning approach. Recently, pruning before training where the architecture is adequately initialized and designed shows that the network utilizing this method can achieve the same accuracy as an entire network [73, 127].

2.4 Weight Sharing

One of the first attempts to reduce network size is associated with layer weight sharing. One key point of this method is that the number of weights that should be shared is not always clear before it starts affecting the overall performance and accuracy of the network. To overcome this issue, some recent works attempted to apply the technique after the initial training instead of prior to training [22, 36, 211]. Nowlan proposed a different approach and Hinton [149], where the authors tried a Gaussian mixture to assign the weights.

An extension of this work was provided later by Ullrich et al. [211], where the optimization method was based on soft-weight sharing. A different approach was seen in the works of Zhang et al. [231] and Plummer et al. [162], where they tried to learn which weights or parameters must be shared. Parameter hashing is another option where parameters are grouped to share weights randomly [36] or share weights of the same value [187, 223]. Another option for weight sharing is using transformers, as Dabre and Fujita [46] and Xiao et al. [227] proposed. Bai et al. [22] proposed deep equilibrium models to find the network's point where backpropagation can be utilized. One more type of parameter hashing is when recursively reusing layers by feeding again into the input the result of the output layer [64, 110, 178]. The work of Kim et al. [109] showcased the usage of residual connection among the output and input layers. Tai et al. [202] proposed an extension of this work, where the authors used an element-wise addition regarding the intermediate outputs before passing the result to the final layer. Relevant works that seek to overcome Vanishing or Exploding Gradients (VEGs) issues were proposed by Zhang et al. [233] and Guo et al. [84].

2.5 Neural architecture search

The Neural Architecture Search (NAS) is another method utilized by data scientists who attempt to build NNs under stringent limitations, ready to be implemented in MCUs. It is a process used to select the model with the highest possible accuracy from a predefined space of CNNs [65]. The typical procedure of a NAS system includes the search algorithm, the search space, and an evaluation strategy [65]. According to Heim et al. [96], NAS techniques could be divided into four different categories: hardware and software co-design, hardware-aware and usage of perceptible metrics, no hardware influence, and finally, the usage of proxies characterized in advance.

Fedorov et al. introduced a NAS method, SpArSe, capable of finding dense and sparse CNNs for MCUs [70]. Another NAS with the name TinyNAS was proposed by Lin et al. [65]. This method includes a search approach that is used to optimize the search space in order to fit constrained hardware and then continues with the optimization for the search space. One more example was proposed by Heim et al. [96], where their proposed method utilizes generic and hardware-based optimizations to decrease the memory requirements of NNs and speed up the inference latency.

2.6 Hardware-based Optimizations

Despite software-oriented optimizations, hardware acceleration is also employed when specialized hardware could be optimized to improve speed and parallel processing [38]. The main focus of hardware accelerators is accelerating mathematical matrix operations [186].

The authors, in [198], presented a block-based SC architecture focusing on improvements in energy and latency by dividing inputs into blocks that will be executed next in parallel. The experiments show significant savings in power consumption while retaining high accuracy.

The researchers of [39], proposed PRIME framework where NN applications could be accelerated by utilizing RAM due to its efficiency and potential to perform matrix-vector multiplications. With this architecture and the design provided by the authors, the overall performance is enhanced by 2360x while the consumption is decreased by 895x.

Another team [122] introduced SmartShuttle for off-chip memory accesses to run as an accelerator for DL applications. The scheme can switch among different data reuse schemes and achieve a match for different layers dynamically. The results revealed a better performance than other state-of-the-art approaches, such as Eyeriss [37], which is also an accelerator for CNNs and can optimize energy efficiency by including a hardware accelerator and a chip for Dynamic RAM (DRAM).

One more proposal regarding hardware can be read in work [145], where the authors try to overcome three main design issues: data access, energy consumption, and data movement. Those three key components, especially the third one, data movement, can result in difficulties with bandwidth and latency. Their main focus is to bring computation closer to data by utilizing a technique called Processing In Memory (PIM) [74]. By implementing this method, the overall data movement between memory and the computation units could be reduced or eliminated by a great margin.

Exploiting FPGAs acceleration for matrix multiplications, the authors in [160] presented an implementation of the transformer model and proper hardware scheduling for DL models utilizing also the method of weight pruning. The results reveal low latency regarding the inference on the hardware with speeds up to 10.96 times compared to CPUs and 2.08 when compared to GPUs. Some more examples related to hardware acceleration designs and architectures can be studied in [105, 163, 228].

2.7 Frameworks, libraries, tools, and other techniques

As mentioned earlier, another popular way for training, optimizing, implementing, or even all the procedures could be achieved by utilizing frameworks, libraries, and other toolkits. Software-based solutions such as Google's

TensorFlow (TF) Lite for MCUs [49] and Microsoft's Embedded Learning Library (ELL) [79] enable the design and deployment of ML models on power-efficient devices and even single-board computers such as Arduino and Raspberry Pi [165]. Additionally, the research community has already developed several software-based solutions, such as libraries and diverse sets of tools, for automatically converting pre-trained AI algorithms, like NNs and ML models, and integrating the generated optimized code into researchers' boards. A more detailed overview of the software-based solutions exploited to implement models into MCUs will be provided in Section 4. The infrastructure of the Laboratory and datasets provided by the project "ParICT-CENG" were leveraged to evaluate the frameworks described in this work. In future development, benchmarking results will be presented.

Figure 1 showcases a summary of the technology of TinyML, with information regarding the technology's main optimization methods, the main benefits and challenges, a framework example, some of the most valuable educational resources, and finally, three different boards fully compatible with TinyML.

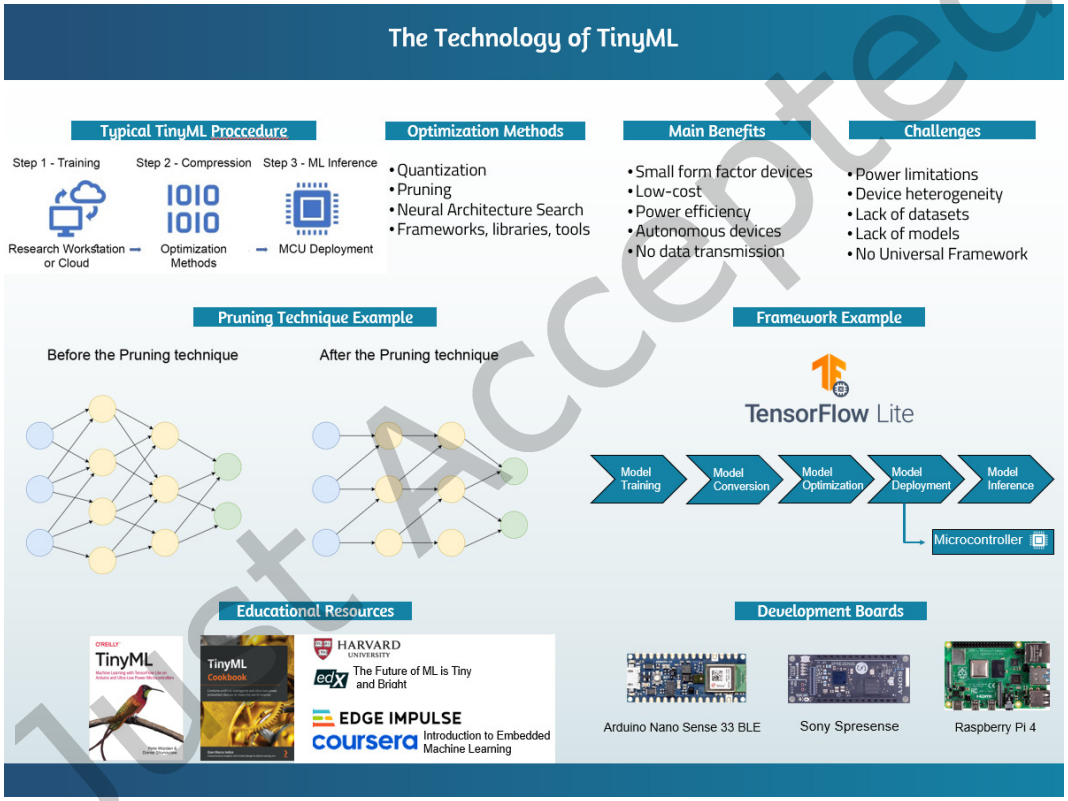

Fig. 1. The Technology of TinyML.

3 TINYML IMPACT

TinyML has witnessed a significant increase in research interest and output in recent years. In the following paragraphs, a bibliometric analysis of the rising number of TinyML research publications from 2019 to 2022, demonstrating the expanding influence of this field, will be provided [1].

Thirteen (13) research papers were published on the nascent subject of TinyML in 2019, indicating initial interest and exploration in this area. The following year, 2020, saw a significant increase in TinyML-related publications, with 88 published papers. This sevenfold increase in just one year demonstrates the rapid recognition of TinyML's potential and applicability in various fields.

In 2021, the number of TinyML papers increased to 346, continuing the upward trajectory. This nearly fourfold increase from the previous year is indicative of the expanding significance of TinyML within the scientific community.

Finally, in 2022, the remarkable amount of TinyML publications reached 724. This doubling of TinyML research output in just one year demonstrates the accelerating rate of innovation and the expansion of TinyML research frontiers.

This bibliometric analysis demonstrates TinyML's rising impact, as evidenced by the exponential growth of research papers over the past four years. The expanding corpus of literature in TinyML highlights the rapid advancements in this field and demonstrates its growing importance in driving the future of intelligent devices.

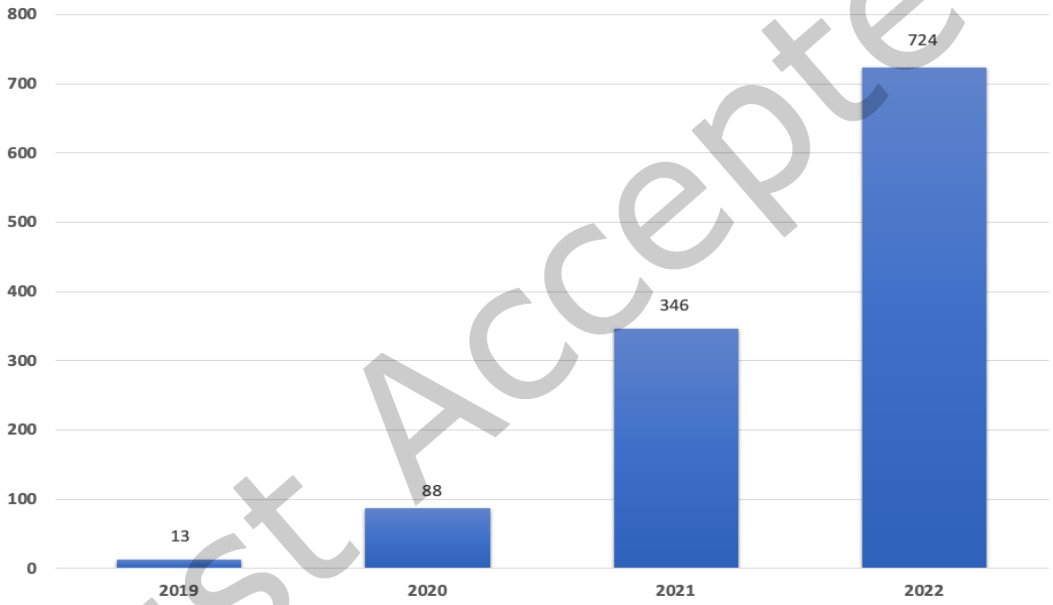

Fig. 2. TinyML Publications.

3.1 Educational materials

Another interesting statistic is that MCUs are currently used in vehicles, consumer electronics, home devices, and industrial equipment. It is expected to have a rise in sales by the end of 2021, followed by higher increases by the year 2023 [101]. This translates to an environment that is mature and prepared to implement the necessary software to create even more innovative devices using TinyML technology. Additionally, one of the most common applications of TinyML is keyword spotting [232]. Many technological champions such as Apple, Google, and Amazon already use a hybrid of keyword spotting applications by inferring ML models into their smart home devices but also depend on the cloud for better results [6, 11, 83]. Another example of utilizing the technology is anomaly detection, a valuable mechanism for security applications [35]. Moreover, since the need for TinyML

operations hardware is publicly available, as stated above, and the recipe for ML applications is very close to the technology of TinyML, a new road for jobs and opportunities is shining. Educational programs for TinyML are extremely necessary to achieve the training mentioned earlier. One of the first collaborations to create educational material was conducted between Harvard University [93] and Google [82] on the edX platform [61]. The course aims to train and prepare students on ways to overcome many challenges, such as the insufficient hardware resources required to run the ML models, and close the growing gap between industry and academia due to the industry's rapid progress. The course teaches the fundamentals of TinyML technology, trains students on real-world TinyML applications, and also informs them about ethical and life-cycle considerations regarding development and deployment.

Additionally, a TinyML kit was co-designed with Arduino [16] to offer low-cost project-based learning to students. The kit contains the Arduino Nano 33 BLE Sense [14], a camera module [13], and a TinyML shield for simplified sensor integration [62, 170]. Another worth-mentioning course is hosted on the Coursera learning platform [43]. The course was developed with some of the most valuable key actors of the TinyML sector: Edge Impulse [60], Arm [20], Arduino [16], and the TinyML Foundation [206]. The course starts with an introduction to ML and embedded systems, continues with NNs and how to train them, and concludes with teaching how to deploy those networks to MCUs. The concepts, demonstrations, and tutorials can be done with the user's smartphone device or Arduino Nano 33 BLE Sense board [44, 59]. Warden and Situnayake offer another great educational resource in the form of a book. The authors introduce ML and embedded systems, while the rest of the book enables the reader to create a series of TinyML projects. The book is ideal for hardware or software developers who want to infer ML models in MCUs [207].

In conclusion, TinyML is the miniaturization of intelligence machinery that can collect data, process it, and extract valuable information in a safe, energy-efficient, reliable, and low-cost way; and has applications in most scientific and industry sectors. There is still significant interest in improving NN capabilities and accuracy [51, 131, 168]. However, with the advances and opportunities TinyML is capable of offering, it is evident that there is now a need to keep accuracy on high levels while also optimizing and compressing the models to be utilized by power constraint hardware, identified as MCUs. Finally, there is a great need to democratize the technology of AI. Nowadays, current AI implementations are developed with businesses and end-consumers in mind and are built to run on research workstations without considering how this could also be achieved in real-world applications and hardware, and even more in constrained hardware. TinyML could bring AI algorithms to more sectors, connect communities, and achieve a digital revolution with more tailored results extracted from data collected near the fountain of each sector and user. As mentioned above, MCUs are among the best-suited hardware devices due to their low cost and worldwide accessibility from everyone [161, 217].

Table 1 provides additional educational material. These educational materials are categorized according to their type, book, course, tutorial, etc. The content the development boards utilize, and their creators are also mentioned.

Table 1. Additional educational materials about TinyML technology.

Name	Type	Content	Development board	Creators
Computer Vision with Embedded Machine Learning [42]	Online course	CV, image classification, deployment, projects, MCUs	Raspberry Pi, Arduino Portenta	Edge Impulse, Coursera, OpenMV, Seed, TINYML

Continuation of Table 1

Name	Type	Content	Development board	Creators
Hello World of AI [181]	Online course	Codecraft, Edge Impulse, projects, data acquisition, hardware, model training, deployment	Wio terminal	Seed, Benjamin Cabé
Everything About TinyML – Basics, Courses, Projects & More! [180]	Collection	Basics, courses, projects	Arduino Nano RP2040, Wio terminal, Seeduino XIAO	Seed, Jonathan Tan
TinyML Study Group [80]	Collection	Basics, optimization, hardware, research paper reading	Neural Compute Stick, Edge TPU board	Archana Vaidheeswaran, Soham Chatterjee
TinyML Example: Anomaly Detection [77]	Tutorial	Data collection, training, Python, ML models, anomaly detection	Adafruit Feather ESP32	Shawn Hymel
Handwriting Recognition [88]	Tutorial	Model training, hardware, evaluation, inference	SparkFun 9DoF	Naveen
CurrentSense-TinyML [78]	Tutorial	Python, Tensorflow, MCU behavior detecting	Arduino Nano 33 BLE Sense	Daniel Cuthbert, Thomas Roth, Mark C.
TapLock - A bike lock with machine learning [89]	Tutorial	Project, MCUs, Edge Impulse, applications	Arduino Nano 33 BLE Sense	Team TapLock
Building a TinyML Application with TF Micro and SensiML [204]	Tutorial	Sensors' utilization, data collection, hardware, deployment	Arduino Nano 33 BLE Sense	Chris Knorowski
Easy TinyML on ESP32 and Arduino [87]	Tutorial	Project, Python, MCU, libraries, deployment	Arduino Nano 33, SparkFun Edge or STM32F746G discovery kit	Eloquent Arduino
Cough Detection with TinyML on Arduino [17]	Tutorial	Data collection, impulse creation, training, deployment	Arduino Nano 33 BLE Sense	UN, Hackster, Edge Impulse
AI Speech Recognition [41]	Tutorial	ML on MCUs, compile, deployment, Python	SparkFun Edge	Dansitu, Mnatraj
TinyML Cookbook [102]	Book	ML on MCUs, Arduino, applications, Python	Arduino Nano 33 BLE Sense, Raspberry Pi Pico	Gian Marco Iodice

4 HARDWARE AND SOFTWARE IMPLEMENTATIONS WITH TINYML

4.1 TinyML-based software

TinyML offers tangible solutions to the urgent energy consumption problems and computational limitations plaguing conventional ML deployment. TinyML employs tiny, energy-efficient processors to perform complex computations on the device itself, unlike conventional methods that often require powerful servers and significant energy resources. This eliminates the need for constant communication with central servers, thereby decreasing latency and energy consumption. Additionally, sensitive data can remain on the device, allowing for increased privacy and security. TinyML represents a significant paradigm shift in implementing and deploying ML, bridging the divide between cutting-edge technology and practical, real-world applications. To achieve this, various frameworks and techniques have been employed to allow the deployment of ML models on constrained hardware devices, specifically MCUs. These tools seek to optimize and efficiently execute ML models, considering these devices' limited memory and power. The frameworks include model conversion, inference execution, hardware-specific optimizations, and support for multiple DL frameworks. Figure 3 is an infographic that visually classifies and illustrates these components of the TinyML ecosystem, providing a comprehensive overview of the various categories and frameworks. The available software can be categorized according to the primary functionality or purpose of each framework/tool as follows:

4.1.1 All in one Frameworks. This subsection explores various frameworks designed for training, optimizing, and deploying ML models to MCUs, enabling embedded devices to perform deep learning tasks within specific memory constraints.

TensorFlow Lite for Microcontrollers is a port of Google's TensorFlow Lite developed specifically for running DL on embedded devices with a few kilobytes of memory, hence significantly expanding the scope of ML.

The Embedded Learning Library (ELL) enables the development and deployment of ML models on platforms with limited resources. A prerequisite is a computer with sufficient resources to generate the machine code the embedded device will execute.

Edge Impulse [60] is a cloud-based framework that offers a complete pipeline to integrate ML models on MCUs. For starters, it allows the user to collect data directly from the device, label them, and create a dataset on the cloud. Edge Impulse has prebuilt ML blocks that can be trained on the provided dataset or even create new ML blocks based on the project's needs. The user has the ability to test the model's performance on a virtual simulation before deploying the model on a physical device. Moreover, while evaluating the model's performance on devices, users can access live classification data to check if the model behaves as expected in real time. Another great feature of this framework is the versatile support of boards and MCUs.

The NXP® eIQ® [152] ML software development environment supports the implementation of ML algorithms on NXP EdgeVerseTM MCUs and microprocessors, including i.MX RT crossover MCUs and i.MX family application processors. The eIQ ML software suite contains the eIQ Toolkit, an ML workflow tool, as well as inference engines, NN compilers, and optimized libraries. The eIQ Toolkit offers graph-level profiling with runtime insights to optimize NN topologies and simplifies ML development using the eIQ Portal and command-line host tools. The eIQ Portal streamlines ML development by providing an intuitive Graphical User Interface (GUI) that allows users to create, optimize, debug, convert, and export ML models.

A comprehensive, all-encompassing framework emerges as the best option for researchers making their initial forays into tinyML and edge devices. The primary justification for this recommendation is the framework's streamlined efficiency: researchers are equipped with a single tool that integrates their models' training, optimization, and deployment processes. Inspecting these frameworks in greater detail reveals two distinct categories: code-based and UI-based Frameworks.

TFLM necessitates that users possess the necessary coding skills, as each phase, from initial model construction to final deployment, must be carried out programmatically. In contrast, frameworks such as Edge Impulse and NXP simplify the procedure and provide a more user-friendly approach. With these platforms, researchers can navigate multiple stages, from training to deployment, with a series of mouse clicks. Such intuitive interfaces can significantly reduce the technical barriers typically encountered in the early stages of research, enabling a more inclusive exploration of tinyML on-edge devices.

4.1.2 Model Conversion. This section focuses on the frameworks and tools that facilitate the conversion of previously trained models into formats suitable for constrained hardware, addressing the gap between complex ML algorithms and the limited resources available on embedded systems.

AIfES (Artificial Intelligence for Embedded Systems) is an open-source artificial intelligence (AI) software framework that can aid in the training and deployment of artificial neural networks (ANN) on a broad variety of hardware. It was created as a

Maker project at the Fraunhofer Institute for Microelectronic Circuits and Systems IMS. Comparable to and compatible with popular Python ML frameworks such as TensorFlow and PyTorch, among others. Feedforward Neural Networks (FNN) are supported in the current release, and it is proposed that Convolutional Neural Networks (ConvNet) will also be implemented in the near future.

Tinymlgen is a Python library that allows TensorFlow models to be exported to C format with minimal code requirements. It accepts a TensorFlow model and returns the necessary C code to be integrated into an Arduino sketch.

Sklearn-porter is another tool capable of converting scikit-learn estimators to C, Java and Javascript among others in order to be utilized in embedded systems. In the same logic m2cgen, a lightweight library used to transpile trained ML models into code to be deployed to constrained devices. Additionally, another tool, weka-porter, is used to transpile trained models from Weka to ready to be deployed code.

Python-based EmbML converts off-board-trained models into C++ or even C source code that can be then compiled and executed on low-power microcontrollers. The primary objective of EmbML is to generate classifier source codes that run specifically on hardware systems with limited resources, using bare metal programming.

FANN-on-MCU is a toolkit based on the Fast Artificial Neural Network (FANN) library for deploying efficient NNs on MCUs based on the ARM Cortex-M series as well as the novel RISC-V-based Parallel Ultra-Low-Power (PULP) platform.

Apache TVM [10] is an open-source compiler framework, that optimizes existing and new ML models to any hardware platform. The two main features of this framework are the compilation of the model to the minimum deployable modules and the optimization of more backend runtimes with better performance. As it is a compiling framework, the supported platforms include CPUs, GPUs, MCUs, FPGAs, and more. Furthermore, this framework is flexible and adaptable to various performance techniques such as quantization and memory planning, optimizing the compiled model to be used on as many platforms as possible.

ScaleDown [179] is an open-source platform for optimizing and deploying NN models to TinyML devices. This is accomplished by providing a framework-independent API that supports widely used DL frameworks. The framework is under development, and new features are constantly added. The optimization techniques that are being offered are pruning, quantization, and knowledge distillation. Additionally, the authors claim to offer conversion of models between different frameworks such as TF, PyTorch, OpenVivo, and ONNX.

AIfES, Tinymlgen, Sklearn-porter, and Apache TVM are crucial in bridging the gap between advanced ML models and hardware environments with limited resources. These tools may be proven useful in areas such as automation, robotics, mobile applications, industrial control systems, and energy-efficient computing. They have facilitated the deployment of neural networks and ML models on a wide range of platforms, from Arduino devices to microcontrollers based on the ARM architecture. Essentially, they encourage innovation and facilitate the rapid integration of ML capabilities into embedded systems with limited resources.

4.1.3 Hardware Accelerators and Hardware Specific Tools. This section examines a variety of TinyML-specific hardware accelerators and tools, focusing on optimization techniques that cater to constrained hardware requirements and enable the deployment of ML models on edge devices.

Arm NN is an ML inference engine for Android and Linux that accelerates ML on Arm Cortex-A CPUs and Arm Mali GPUs by leveraging a number of Arm architecture-related optimizations. This ML inference engine is an open-source software development kit that fills the difference between current NN frameworks and energy-efficient Arm IP.

CMSIS NN software library is a compilation of efficient NN kernels designed to optimize the performance and memory requirements of NNs on Cortex-M processors. Included among the core functions of the aforementioned library are the Convolution Functions, Activation Functions, Softmax Functions, and Basic Math Functions.

In STM32 microcontroller platforms, NanoEdge AI Studio enables the deployment of ML models with on-device learning capabilities. Following step-by-step instructions to collect, validate, and generate the C-code to be incorporated into the final project enables the development of multiple applications, including anomaly detection and classification.

X-CUBE-AI is an STM32Cube Expansion Package that contributes to the STM32Cube.AI ecosystem. It can automatically convert pre-trained models, into STM32CubeMX executables.

HANNAH is a hardware accelerator and NN search framework designed to solve the hardware-software co-design prerequisite of TinyML and meet constrained hardware requirements. As the authors of the framework suggest [29], HANNAH aims to automate the training, deployment, and optimization process of NN architecture and bring intelligent data processing

to edge devices. The NPU architecture that is based on UltraTrail [27] and utilized at the deployment step embodies an array of multiply and accumulate units for convolutional layers and provides analytical models to avoid long times during simulation and synthesis of the hardware. During the optimization process, a joint search space is used to combine the NN and the NPU design space. There is the option to restrict the number of design choices, and the optimization function is assembled by metrics such as the estimated power and latency.

Hls4ml [68] is an open-source framework that provides software and hardware co-design for ML algorithms targeting FPGA devices and ASIC technology. The framework's workflow can automatically translate NNs with specifications such as the model's architecture and weights into a hardware accelerator ready to be processed with High-Level Synthesis (HLS) tools. The network can be designed with (Q)Keras and PyTorch before being translated into an HLS project. The workflow also includes quantization and pruning aware training for the essential optimization to prepare the network for device implementation.

As presented in [201], the PULP framework can run non-neural ML algorithms such as K-means, KNN, and SVM into the constrained hardware of Parallel UltraLow-Power (PULP) MCUs. The authors developed the system to target the commercial chip GAP8 and the research platform PULP-OPEN, which is able to run on FPGA emulators. The main focus of the framework in consideration is to achieve the same level of performance when compared to NNs due to the parallel design of the algorithms. This design allows speedups up to 7.64x. Finally, the 8-core clusters of the PULP-OPEN platform can achieve up to 15.85x improvement regarding performance compared to Cortex-M4.

Accelerators and tools, such as Arm NN, CMSIS NN, NanoEdge AI Studio, and HANNAH, are revolutionizing the deployment of ML models on peripheral devices. From real-time gesture recognition in wearables to anomaly detection in industrial machinery, the aforementioned tools provide optimization techniques tailored to constrained hardware requirements for applications such as real-time gesture recognition in wearables and anomaly detection in industrial machinery. They have contributed to the expansion of TinyML, which has enabled the efficient execution of DL algorithms on platforms such as Android devices, Cortex-M processors, STM32 microcontrollers, and FPGA devices. By aligning with the demands for energy efficiency and real-time processing, they are establishing a new standard for intelligent data processing at the edge by creating novel possibilities for ML applications in scenarios where hardware with limited resources is required.

4.1.4 Tools for deployment. This section describes the specialized tools that facilitate the deployment of ML models on microcontrollers and embedded devices, expediting the deployment process and ensuring that models are executed efficiently on constrained platforms.

emlearn is an inference engine for Machine Learning on Microcontrollers and Embedded Devices. The tool enables the training of the model using scikit-learn or Keras and then generates a portable C99 code that can be deployed to constrained devices with a simple header file include.

uTensor is a Tensorflow-based, Arm-optimized ML inference framework with a minimal footprint. It comprises a runtime library and an offline utility that performs most of model translation work.

DORY [31] is a framework for automatically deploying DNNs in low-cost MCUs with less than 1MB of available SRAM. The framework maximizes L1 memory use within the limitations given by each DNN layer. It generates ANSI C code to manage transfers and calculation phases. Authors' experiments on GreenWaves technologies GAP8 revealed that DORY outperforms the GreenWaves solution by up to 2.5 MAC/cycle and the result on an STM32-H743 MCU by 18.1 MAC/cycle. GAP-8, when utilizing the framework in consideration, can achieve end-to-end inference of a 1.0-MobileNet-128 network for an average of 63 pJ/MAC @ 4.3 frame per second, which translates into 15.4 times faster than an STM32-H743.

The aforementioned tools are primarily designed for developers with higher expertise, especially those who have already trained and optimized their ML models. In addition, for professionals and researchers who choose well-known libraries, such as Scikit-learn or Keras, for their model training endeavors, the second-mentioned tool appears to be particularly appropriate. Emlearn has received much attention and appears to be establishing itself as a preferred option among the academic community. This claim is supported by the growing number of scholarly works that cite or employ Emlearn in their methodologies and findings.

4.1.5 Tools for optimization. This section explores the open-source libraries and frameworks that provide optimization techniques for deploying quantized neural networks on MCUs. It highlights innovations that reduce computational and resource costs, making them ideal for hardware environments with limited resources.

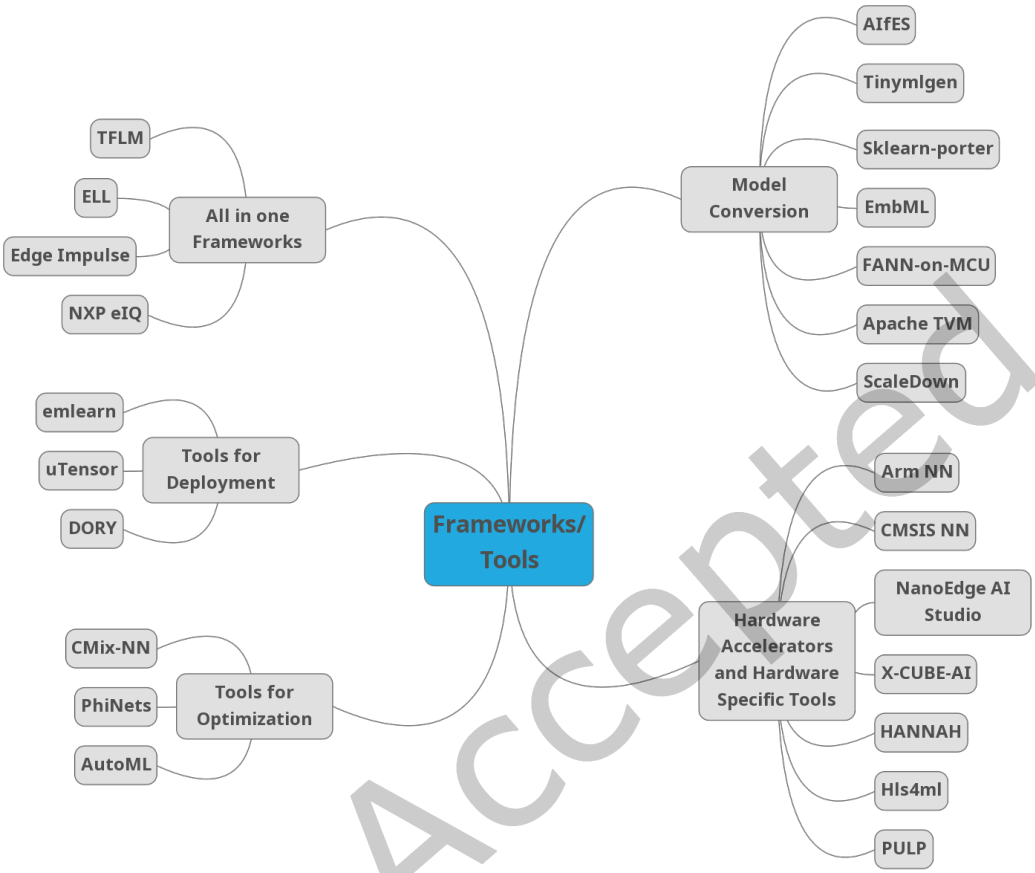

Fig. 3. Frameworks and Tools for ML Deployment on MCUs

CMix-NN is an open-source mixed-precision library for MCU deployment of quantized neural networks. The library supports convolutional kernels with any bit precision* in the range of 8, 4, and 2 bits for any of the convolution operands.

PhiNets [155] is a scalable framework based on residual blocks designed to reduce the computational and resource cost for image processing on constrained hardware. The framework builds the network with a sequence of inverted blocks where each one is followed by a swish activation function. Additionally, the first layer is represented by 24a, where, a, is a hyperparameter, and the multiplication factor is doubled each time the feature map is downsampled. After a convolutional block, Squeeze-and-Excitation blocks [99] are inserted, and skip connections are utilized among the same resolution bottleneck layers. Across the network, five stridden convolutions are implemented for the required down-sampling of the feature maps with a reduction of a factor of 32x between the input and output tensor. In the last convolutional block a neck of a 2x up-sampling layer exists, and a skip connection to help with the performance degradation.

AutoML [192] is a framework that utilizes hybrid-block structured and Pattern Pruning (PP) to enable the efficient execution and reconfiguration of NLP models based on transformers on constrained hardware. This reconfigurability is critical for energy savings in battery-powered devices, which frequently employ the Dynamic Voltage and Frequency Scaling (DVFS) technique for hardware reconfiguration in order to extend battery life. The optimization strategy comes on two levels. First, it

compresses the data using an efficient Block Pattern (BP) and then heuristically shrinks the search space based on the initial findings.

As described in Section 2, where various optimization techniques are presented, the libraries under consideration have exhibited remarkable proficiency. Specifically, they have been able to make significant progress in the field of model and neural network (NN) compression. These enhancements are crucial because they facilitate the deployment of these models and NNs on edge devices, which are frequently characterized by severe resource limitations. This capability illustrates their potential to facilitate the wider adoption of ML solutions in environments with limited resources.

Moreover, it is worth mentioning that one of the first attempts at online learning and on-device training already exists. TinyOL (TinyML with Online-Learning) permits incremental training on-device for streaming data. TinyOL is founded on the concept of online learning and is appropriate for IoT devices with limited resources.

Table 2 provides a summary of the aforementioned frameworks, tools, and libraries, including the algorithms and platforms utilized by each software, the supported programming languages and compatible libraries, as well as the software's creators and open source status.

Table 2. TinyML software.

Name	Algorithm	Platforms	Languages	Libraries	Applications	Open Source	Creator	Type
TensorFlow Lite	NN	Cortex-M, Cadence Tensilica	C++, Java, Python, Swift, Objective-C,	TF, TF Hub	image & text classification, object detection, pose estimation, question answering	Yes	Google	Framework
ELL	NN	Cortex-M, Cortex-A micro:bit	C++, Python	CNTK, ONNX, Darknet	image & audio classification	Yes	Microsoft	Framework
ARM-NN	NN	Cortex-A, Mali GPU, Ethos NPU	C	TF Lite, ONNX	variety of applications	Yes	ARM	Tool
CMSIS-NN	NN	Cortex-M	C	TF, Caffe, PyTorch	N/A	Yes	ARM	Library
NanoEdge AI Studio	NN	Cortex-M	C	N/A	anomaly & outlier detection, classification, regression	No	STMicroelectronics	Framework
X-Cube AI	NN, classic ML	STM32	C	Keras, Caffe, ConvnetJS, Lasagne, ONNX	isolation forest, SVM, K-means, etc.	No	STMicroelectronics	Library-Tool
AIfES	FNN	Windows (DLL), Raspberry Pi, Arduino UNO, Nano 33 BLE Sense & Portenta H7, STM32 F4, ATMega32U4	C++, C#, Python, Java, VB.NET	TF, Keras, PyTorch	IoT sensors, medical wearables, smart environments, condition monitoring	Yes	Fraunhofer IMS	Framework
TinyMLgen	NN	Arduino Nano 33 BLE Sense, STM32, SparkFun Edge, ESP32	C	TF Lite	variety of applications	Yes	Individual	Library-Tool

Name	Algorithm	Platforms	Languages	Libraries	Applications	Open Source	Creator	Type
MicroMLGen	Decision Tree, Random Forest, XGBoost, SEFR, GaussianNB, SVC, OneClassSVM, Relevant Vector Machines, PCA	Arduino Uno, Nano, Micro, etc., ESP32, any MCU supporting C	Python	SciKit-learn	classifiers	Yes	Individual	Library-Tool
emlearn	Decision trees, NNs, Naive Gaussian Bayes, Random Forest	AVR Atmega, ESP8266, ESP32, Cortex-M	Python	SciKit-learn, Keras	classifiers, outlier & anomaly detection	Yes	Individual	Library-Tool
sklearn-porter	Decision trees, NNs, Naive Gaussian Bayes	constrained platforms	Java, JS, C, Ruby, Go, PHP	SciKit-learn	classifiers, regression	Yes	Individual	Library-Tool
m2cgen	Linear & Logistic regression, NNs, SVM, Decision tree, Random Forest, LGBM, classifiers	constrained platforms	C, C#, F#, Dart, Go, Haskell, Java, JavaScript, R, PHP, Python, Ruby, Rust, Visual Basic	SciKit-learn	classification, regression	Yes	Individual	Library-Tool
weka-porter	Decision trees	constrained platforms	C, Java, Javascript	weka	classification	Yes	Individual	Library-Tool
EmbML	Logistic regression, Decision trees, MLP, SVM	Arduino, Cortex-M4	C++, Java, Javascript	SciKit-learn, weka	classification	Yes	Research group	Library-Tool
uTensor	NN	Arduino, Cortex-M4	C++	TF	N/A	Yes	Individual	Framework
TinyOL	NN	Cortex-M	C++	any NN	N/A	No	Siemens - Research group	Framework
FANN-on-MCU	NN	Cortex-M, RISC-V	C	FANN	variety of applications	Yes	Research group	Library-Tool

Name	Algorithm	Platforms	Languages	Libraries	Applications	Open Source	Creator	Type
CMix-NN	NN	Cortex-M	C	Mobilenet	N/A	Yes	Research group	Library-Tool
Edge Impulse	NN	mcu, CPU, accelerators	C++	N/A	Anomaly detection, classification	No	Edge Impulse	Framework
Apache TVM	NN	CPU, FPGA, GPU, MCU	C++, Rust, Java	Keras, CoreML, PyTorch, TF, MXNet, DarkNet	variety of applications	Yes	Apache	Framework
NNTOOL	NN	Cortex-M, Cortex-A	C, C++	TFLite, ONNX, DeepView RT	Classification	No	NXP	Library-Tool
DORY	NN	Cortex-M, STM32	ANSI C	Mobilenet	classification	Yes	Research group	Framework
AutoML RT	NN	N/A	N/A	N/A	NLP	N/A	Research group	Framework
ScaleDown	NN	Constrained platforms	Python	TF, PyTorch, ONNX, OpenVivo	classification	Yes	Research group	Library
PULP	ML algorithms	PULP-OPEN hardware	C	N/A	N/A	Yes	Research group	Library
hls4ml	NN	FPGA	Python	Keras, Onnx, TF, QKeras, PyTorch	N/A	Yes	Individual	Library
PhiNets	NN	Constrained platforms	Python	PyTorch, Keras	image & signal processing	Yes	Research group	Framework
HANNAH	NN	Constrained platforms	C	N/A	wearable healthcare devices, key-word spotting	N/A	Research group	Framework
OctoML	NN	ARM, Xilinx, NVIDIA, Intel, Qualcomm	C	N/A	NLP, CV	No	OctoML	Framework

4.2 TinyML-based hardware and development boards

The advantages of embedding ML models on hardware are undeniably numerous, and as mentioned earlier, security, instant, tailored results, and the creation of autonomous devices are some of the most resounding examples. Several researchers are also exploring this new sector and analyzing why the industry should start inferring models on hardware, how it is achievable, and finally, why the technology of ML must become more accessible to everyone [49, 176, 217, 232]. Two of the most popular boards ready to run any type of software, specifically the technology in consideration, are Raspberry Pi and Jetson Nano from Nvidia [151]. Jetson is a great example of a tiny computer able to fit in a palm. It is powerful and most commonly used in automotive and robotics. In those two sectors, any application built requires a vast amount of power source, which can also power the demanding board from Nvidia. For smaller, innovative, and not that demanding applications such as wearable devices, the need for an external power supply makes using the board prohibitive [222]. The solution is the use of another constrained hardware, the MCU. An MCU is a hardware platform that operates below 1 mWatt and has a few kilobytes of RAM, typically with the same amount of flash memory for data storage. Nowadays, most MCUs switched to 32-bit CPUs thanks to ARM. To be able to operate at the cost of a single small battery, MCUs do not always have an operating system, are single-threaded, and avoid the usage of dynamic location functions [24, 72, 199, 222]. With the advances in technology and by understanding the need for more ingenious IoT devices, intelligent machines, and wearable health-related devices, many manufacturers have developed small-factor and low-cost boards specifically designed for ML inference.

A popular board is from Sparkfun called Edge, utilizing an Ambiq Apollo3 MCU, the MCU's schematic is visualized in Figure 4, which is powered by Arm Cortex-M4, operates at 48 MHZ, and has a TurboSPOT burst mode for reaching 96 MHZ. The board requires an ultra-low supply current at 6 μA/MHz executing from flash at 3.3 V; it has a flash memory of 1 MB and up to 384 KB of RAM [7]. Nano 33 BLE Sense [14] is the board of preference for many educational courses since it has a unique TinyML Kit equipped with components and hardware required for students to start embedding ML models and also used for research and small projects [18, 19, 48, 170, 172].

Table 3 summarizes the development boards mentioned above, as well as additional boards that use TinyML technology. The type of MCU and CPU, the CPU clock, the capacity of RAM and flash memory, and the dimensions of each board are reported.

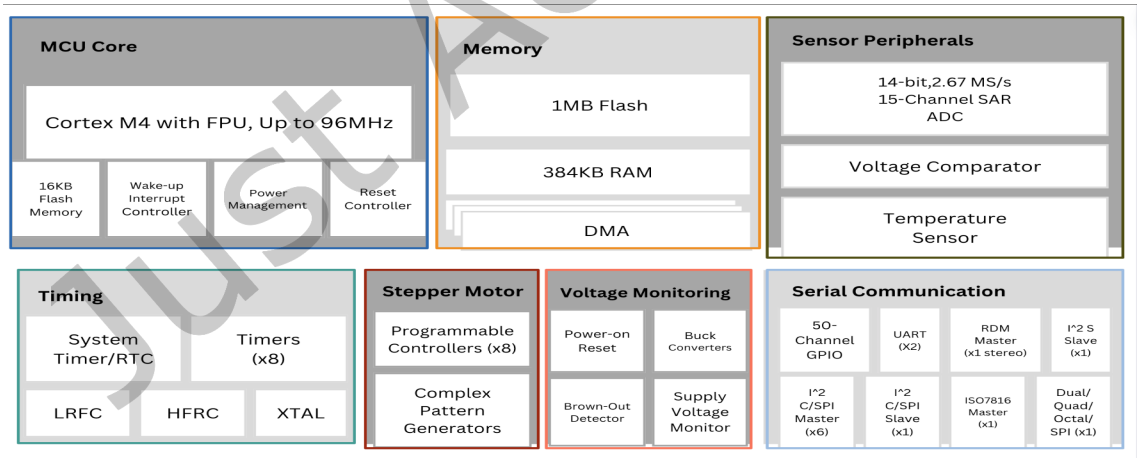

Fig. 4. Ambiq Apollo3's schematic

Table 3. Development boards that support TinyML.

Board	MCU	CPU	CPU Clock	Flash memory	SRAM	Dimensions	Applications
Seeeduino XIAO [182]	SAMD21G18	ARM Cortex-M0+	up to 48 MHz	256 KB	32 KB	20x17.5x3.5 mm	wearable devices, rapid prototyping
B-L475E-IOT01A Discovery kit [196]	STM32L4	ARM Cortex-M4	80 MHz	1 MB	128 KB	61x89x9 mm	applications with direct connection to the cloud
Syntiant TinyML [200]	NDP101	Cortex-M0+	48 MHz	256 KB	32 KB	24x28 mm	speech and sensor applications
Arduino Nano 33 BLE Sense [14]	nRF52840	Cortex-M4	64 MHz	256 KB	1 MB	45x18 mm	wake word and motion detection
Portenta H7 [15]	STM32H747	Cortex M7 and Cortex M4	480 MHz and 240 MHz	16 MB NOR Flash	8 MB SDRAM	62x25 mm	computer vision, robotics controller, laboratory equipment
Sony Spresense [193]	CXD5602	ARM Cortex-M4F ×6 cores	156 MHz	8 MB	1.5 MB	50x20.6 mm	sensor analysis, image processing
Raspberry Pi 4 Model B [166]	BCM2711	Quad-core Cortex-A72	1.5 GHz	N/A	2 GB, 4 GB, or 8 GB SDRAM	56.5x86.6 mm	robotics, smart home
Raspberry Pi 4 Pico [167]	RP2040	Dual-core ARM Cortex-M0+	up to 133 MHz	2 MB	264 KB	51x21 mm	wake up words
Jetson Nano [151]	N/A	Quad-core ARM A57	1.43 GHz	N/A	4 GB LPDDR4	70x45 mm	robotics, computer vision
SparkFun Edge [195]	Apollo3	ARM Cortex-M4F	up to 96 MHz	1 MB	384 KB	40.6x40.6 mm	motion sensing
Adafruit EdgeBadge [4]	ATSAMD51J19	ARM Cortex-M4F	120 MHz	512 KB	192 KB	86.3x54.3 mm	image processing
Wio Terminal [183]	ATSAMD51P19	ARM Cortex-M4F	120 MHz	4 MB	192 KB	72x57 mm	remote control, monitoring
Himax WE-I [97]	HX6537-A	ARC 32-bit DSP	400 MHz	2 MB	2 MB	40x40 mm	image processing, voice and ambient sensing
ESP32-S3-DevKitC [67]	ESP32-S3-WROOM-1	32 bit Xtensa dual core	240 MHz	N/A	512 KB	N/A	rapid prototyping
Arducam Pico4ML-BLE [12]	RP2040	Dual-core ARM Cortex-M0+	133 MHz	2 MB	264 KB	51x21 mm	image processing, data collection

5 APPLICATIONS WITH TINYML

This section provides a chronological overview of publications linked to TinyML. These works are classified according to their field of study.

5.1 Healthcare

Paul et al. [159] developed a real-time TinyML-based American sign language system. The proposed system is a highly efficient CNN-based fingerspelling recognition system that embedded TinyML in a Cortex-M7 with a 158 KB size board. The proposed solution reduces quantization's accuracy drop and generalizes the model via interpolation augmentation. TinySpeech [226] are low-precision DNNs that are designed for limited-vocabulary speech recognition. The experimental results of the proposed TinyML NN showed significantly lower architectural and computational complexity compared with other DNNs for limited-vocabulary speech recognition.

The authors of the study [71] implemented recurrent NNs for hearing aid hardware. Several TinyML techniques, such as pruning and integer quantization of weights/activations were utilized.

A wearable system based on regular shoes for foot gesture recognition was implemented by Orfanidis et al. [154]. The system can track the user's specific foot gestures in the environment of a smart city, and when the user is in danger, the system notifies his/her familiars. The overall process was implemented with an embedded NN in an MCU. The evaluating results have shown a 98% accuracy among five different activities and foot gestures.

A wearable TinyML-based wristband for hand gesture recognition was proposed in [28]. The experimental results show that the proposed TinyML-based wristband can recognize seven hand gestures with 96.4% accuracy. Also, a hand gesture recognition approach for wearable devices is presented in [234]. The proposed approach uses accelerometer and electromyogram signals for a dual-stage classification in a memory-efficient way. The TinyML-based approach achieved a 93.34% classification accuracy, a 17.79% improvement of the trained model, and a significant decrease in the device's memory footprint.

A cloud computing system for high-level supporting systems combined with TinyML for prognostics and health management was proposed in [237]. The authors of this work investigate sensor-based applications and predict health status while combining TinyML with cloud computing to adapt the system's diverse requirements, like power, latency, and communication.

A non-invasive TinyML-based system for real-time activity tracking for elderly people and their nurses was proposed by T'Jonck et al. [208]. The system provides real-time elderly concerns information like incontinence, night wandering, and pressure ulcers. Furthermore, a healthcare body-pose estimation platform-agnostic framework was proposed in [215]. The proposed application monitors patients and alerts when they fall off their bed, have an accident, or experience restricted movement.

Ooko et al. [153] used TinyML-based NN models to predict in real-time chronic obstructive pulmonary disease. The proposed approach improves the inference accuracy of portable and non-invasive self-diagnostic kits for respiratory diseases. Also, a low-cost mechanical ventilator, namely A-Vent, was developed by Cabacungan et al. in the study [32]. The proposed ventilator uses TinyML models to detect patient-ventilator asynchrony. The experiment results showed that TinyML could be trained to detect breathing anomalies and provide low-cost and real-time remote monitoring.

Most of the aforementioned healthcare systems reported improved system accuracy because of the use of TinyML. Furthermore, healthcare devices and systems are crucial to providing real-time and tailored results. Finally, due to the local data processing, sensitive and private data are kept secure and private.

5.2 Automotive

In the work [115] a wearable TinyML-based alcohol sensor was proposed. The usage of the proposed wearable system is to detect the consumed level of alcohol. The system's ML model was trained by the sensors' collected data, and the TinyML model was uploaded to an nRF52480 MCU for the prediction of test data to account for the impact of environmental conditions on the alcohol sensor. The results indicate that the method is useful for refining sensor response implementation in a variety of heat and humidity conditions.

The authors of work [174] presented a novel approach for automating traffic scheduling based on the density of vehicles waiting in line. The proposed system detects vehicles with embedded sensors across the road's lanes and a TinyML model was built to predict the green signal timings and to control the traffic system efficiently. In work [148] a TinyML-based sensor for the classification of the vehicle's type and the speed range was designed. The presented sensor is powered by the battery and mounted in the pavement for the vehicles' classification with the use of recurrent NNs. The experimental results show an accuracy of 96% in vehicle's type classification and 89% in vehicle's speed classification.

A TinyML approach for detecting road anomalies (potholes, bumps, and obstacles) on vehicles was proposed in [8]. The unsupervised TinyML technique is embedded in an Arduino Nano BLE 33.

Raza et al. [169] presented a novel approach for more considerable autonomy and intelligence in micro aerial vehicles. This is achieved using TinyML technology via OpenMV for lower latency, energy efficiency, offline inference, and data security in drones. The experimental results of this study revealed that integrating TinyML-based MCU into the drones makes the system viable in a practical context.

The authors of the study [50] presented the deployment of TinyML models for autonomous driving mini-vehicles. The authors, with the use of high-throughput TinyCNNs having access to an on-board camera, control mini-vehicles and succeed in minimizing the inference energy consumption.

In the automotive field, TinyML-based systems succeed in significant improvements in systems energy efficiency which are meaningful advantages in the modern automotive.

5.3 Agriculture

Vuppalapati et al. [216] presented a novel framework for TinyML-based agriculture sensors. The proposed framework exploits Azure DevOps automation techniques and TinyML to provide high-quality products, in the most cost-efficient way, ensuring farmers' cost savings. Also, the study [218] proposed a TinyML-based framework for rural areas and small farmers. The system's TinyML IoT edge devices collect real-time data from a real production environment, contributing to creating a sustainable food future.

The authors of the study [5] presented a real-time embedded prediction weather system. The system was implemented using tiny DNNs in an STM32 MCU to predict real-time environmental conditions. The main advantage of this system, rather than the others that have been proposed, is the performance of prediction close to the environmental sensor to avoid data traffic to the cloud.

Work [209] illustrates an end-to-end strategy for enhancing the security of the food supply chain and, consequently, boosting the credibility of the food sector. The system seeks to increase the transparency of food supply chain monitoring systems by securing their constituent parts. A universal information monitoring strategy based on blockchain technology protects the integrity of gathered data, while a self-sovereign identification approach for all supply chain participants reduces single points of failure. Finally, monitoring devices are fitted with a security mechanism based on TinyML's fledgling technology to mitigate a major amount of harmful supply chain actor activity.

TinyML-based agriculture systems provide farmers with tailored results and autonomous, low-cost systems. Moreover, using TinyML in this area reduces access to cloud services, resulting in low network bandwidth and lower costs.

5.4 Security

Dutta and Kant [57] designed a TinyML-based framework to predict potential threats that propagate to smart devices. The proposed framework copes with the various security challenges in the different protocols' layers of IoT devices. Furthermore, a TinyML-based framework for detecting devices' physical anomalies was proposed in [128]. With the use of an Arduino Nano 33 BLE that embedded DL TinyML models mounted in a washing machine, physical anomalies were detected via a battery-powered embedded device with no network connection.

In the security field, the TinyML approach can shield IoT devices in an automated way, while requiring low power consumption.

5.5 Industry

Acharjee and Deb, in their work [3], used TinyML strategies, such as post-training quantization, to generate cartoonized versions of real-world images. The proposed method's testing results showed that the implemented model with post-training quantization achieves a high compression level. An embedded system for TinyML-based audio classification was presented in [114]. The proposed technique improves the operation speed and decreases the time and the energy consumption. Also, the experiments have proved that co-designing both hardware and software reduces execution overhead when compared to optimizations used only on the software side.

Kamal et al. [106] proposed an architectural design for checking and quality auditing machine objects using a TinyML DNN. The proposed design consists of i) the machine object module, ii) the edge computing unit to connect, configure, and control the module, iii) the edge server that embeds the TinyML, and iv) the cloud services for the quality audit and predictive maintenance of the machine objects. Giordano et al. [76] developed a TinyML-based wireless camera for face recognition. The TinyML algorithm for face recognition was hosted in an ARM Cortex-M4F MCU for the onboard data processing, while the information on recognized faces was sent via a long-range LoRa communication module. A working prototype of the system was evaluated both for the capability of battery-less and self-sustainability, with a high accuracy of up to 97%.

TinyML-based industry systems improve the system's operational speeds while providing tailored and real-time results. Also, as communication with cloud services is nearly eliminated, local data processing provides low power consumption and more privacy.

5.6 Future Directions

Despite the wide variety of TinyML applications present in both research and industrial domains, the technology is not without its challenges. These challenges, which will be elaborated upon in the subsequent section, have been the subject of considerable efforts aimed at their resolution. In the forthcoming paragraph, a succinct overview of various prospective trajectories for TinyML will be presented. These trajectories encompass potential enhancements to the technology itself and ways in which the technology could be employed to augment deployed systems and applications.

(1) Reformable TinyML - Most current applications are based, as work [214] characterizes TinyML in static TinyML. Static TinyML is defined by the one-time model inference, where once the TinyML model is trained and deployed, update via network or further training is non-existent [121]. In the above-mentioned work [214], Reformable TinyML is discussed, which, according to the authors, addresses issues in static TinyML and encompasses solutions capable of updating the models embedded to maximize performance in the environment they are placed in. The approaches mentioned to improve the models include On-device Offline Learning approaches, Online Learning Approaches, and Network Reliant approaches.
(2) Autonomic Computing - The field of autonomic computing [75] is dedicated to exploring how systems can autonomously attain user-specified control outcomes, thus obviating the necessity for human involvement. Control theory has substantially influenced the foundational principles of autonomic computing, particularly in relation to closed- and open-loop systems. Incorporating AI and ML methodologies can facilitate realizing such precise autonomic and self-managed systems. Therefore, the combination of TinyML and DRL models has the potential to accelerate the development and adaptation of autonomic systems across a diverse array of sectors.
(3) Blockchain Integration - In the context of updating or retraining an existing TinyML model via a network, the security attributes and capabilities provided by blockchain technology appear well-suited to address the majority, if not all, potential security threats presented by malicious users. In [40], [113], and [25], the peer-to-peer propagation of firmware updates in IoT devices is investigated. In addition, [203] provides an overview of how blockchain could be adapted to various IoT implementations.
(4) Edge offloading - The provision of sufficient computational capacity for intelligent edge applications has emerged as a formidable obstacle. Intelligent Edge, which propels intelligence to the Internet's Edge, has been instrumental in facilitating intelligent decision-making across multiple aspects of edge computing, such as task offloading [175]. Edge

offloading is a paradigm of distributed computing that provides computing services for edge caching, edge training, and edge inference. Incorporating techniques such as Distributed Machine Learning (DML), DRL, and Collaborative Machine Learning (CML) into edge computing is advantageous for managing the escalating communication and computational demands of emergent IoT applications [187]. Given the findings of these studies, blockchain could serve as a secure method of updating without the possibility of third-party interference or the necessity for local retraining of any TinyML model currently in use.

6 THE CHALLENGES OF TINYML TECHNOLOGY

TinyML is found in the early stage of its development, and since it is a technology meant to be implemented on constrained hardware, the first issue noted is the resource and computational limits of the applications it may face. Additionally, the lack of universal frameworks, device heterogeneity, lack of models, datasets, and benchmarking tools, along with several security issues, are some of the most critical challenges to overcome. A brief description of the aforementioned challenges follows in the next paragraphs.

6.1 Resource and computational constraints

Until now, a typical ML model or NN was meant to be trained and run on powerful research workstations or in the cloud [176]. Implementing and deploying models, though, in a device that not only lacks powerful CPUs and GPUs but, on the contrary, relies on an MCU with limited memory, storage, and CPU capabilities is a different story. The majority of the devices, as mentioned earlier, have an average clock speed of 100 MHz, with less than 1MB of flash memory. As indicated above, optimizing a model to be suitable for deployment in such a device can be a complex process that mostly affects the model's overall accuracy. There is a need for new and more aggressive optimization methods along with advances regarding memory and CPU speeds in the hardware sector.

6.2 Device heterogeneity

Software and hardware heterogeneity can be identified as the second major issue. The diversity of hardware and algorithms makes prohibitive the development of a universal framework capable of training, optimizing, and deploying a model to several different devices with a single compilation. Since every system is different, despite the fact that two different systems may share the same components, a model trained for a specific device may not work to another one [172]. Furthermore, as stated many times, the ecosystem of TinyML requires careful software and hardware co-design. Building models universally could hinder accuracy, power consumption, and storage requirements.

6.3 Lack of datasets, models, and benchmarking tools

The aforementioned issue continues and impacts important principles of ML, such as the development of universal datasets, models, and tools used for testing and benchmarking. Before creating ML applications, the significant first step for data scientists is to acquire the appropriate data from sensors or datasets. There is a vast amount of publicly available datasets, but since the technology is still in an evolving stage, those datasets are not altered to reflect the energy and hardware constraints found on MCUs. The community also must create pre-trained models that require low to moderate changes to be deployed on different devices. Finally, an essential step of the ML procedure is testing and benchmarking. Models and algorithms must be compared and evaluated to get a better knowledge of when and how to implement models for different situations and sectors [194]. MLCommons is one of the first attempts to accelerate ML innovation by providing the correct rules and requirements for benchmarking and datasets, and also provides the best practices to help or, as stated on their site, empower researchers by exchanging models experiments, and applications [141].

6.4 Security

Constrained devices are not built with a focus on security due to their limited resources. Even more complex models and data storage may still force developers to upload some of the data to an external source, server, or the cloud. Data transmission, of course, shares the same dangers found on simple IoT devices that depend on external sources for processing demanding operations such as AI and NN applications.

7 CONCLUSION

This work provides a brief review of the most common optimization techniques that led to the development of TinyML. Furthermore, due to the fact that TinyML is a hybrid of hardware and software, a taxonomy of the development boards as well as the required frameworks, tools, and libraries is presented. Finally, this work includes educational resources for the technology discussed as well as a brief overview of TinyML-based applications organized into five categories. Finally, the future directions regarding the technology under consideration are presented. This study seeks to demonstrate the benefits of TinyML technology and provide useful information to anyone interested in researching and working on the subject.

Incorporating ML into tiny, resource-constrained embedded devices is becoming increasingly important in light of future applications. Improving the current context of ML at the edge devices, TinyML has been introduced as a way to build autonomous and secure devices that can gather, process, and provide results or decisions without having to share data with third parties. TinyML can be integrated into low-cost, low-power smart devices such as smartphones, microcontrollers, and IoT-edge systems. The presented technology intends to democratize AI by making it available to a wider variety of industries and communities, allowing everyone to participate in the digital revolution of intelligent devices.

Although TinyML technology has many applications in a variety of fields and revolutionizes the thought process and democratization of ML and AI applications, it is introducing many challenges for researchers to unlock its full potential and embed new features. TinyML is in its early stages of development, and being a technology intended for implementation on constrained hardware, the first issue identified is the resource and computational limitations that applications may encounter. In addition, the absence of universal frameworks, device heterogeneity, lack of models, datasets, and benchmarking tools, as well as various security vulnerabilities, are among the most difficult obstacles to surmount.

ACKNOWLEDGMENTS

We would like to thank Georgios Giannakas for assisting us in the research and the preparation of this work. Also, we acknowledge support of this work by the project “ParICT_CENG: Enhancing ICT research infrastructure in Central Greece to enable processing of Big data from sensor stream, multimedia content, and complex mathematical modeling and simulations” (MIS 5047244) which is implemented under the Action “Reinforcement of the Research and Innovation Infrastructure”, funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

REFERENCES

[1] [n.d.]. TinyML in Publications - Dimensions. https://app.dimensions.ai/discover/publication?search_mode=content&search_text=TinyML&search_type=kws&search_field=full_search

[2] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. 2017. Mobile edge computing: A survey. *IEEE Internet of Things Journal* 5, 1 (2017), 450–465.

[3] Jashaswimalya Acharjee and Suman Deb. 2021. Cartoonize Images using TinyML Strategies with Transfer Learning. In *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*. IEEE, 411–417.

[4] Adafruit 2021. *Adafruit EdgeBadge - TensorFlow Lite for Microcontrollers.* Retrieved December 30, 2021 from https://www.adafruit.com/product/4400

[5] Francesco Alongi, Nicolo Ghielmetti, Danilo Pau, Federico Terraneo, and William Fornaciari. 2020. Tiny Neural Networks for Environmental Predictions: an integrated approach with Miosix. In *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 350–355.

[6] Amazon 2022. *Amazon Alexa.* Retrieved January 8, 2022 from https://www.amazon.com/b?ie=UTF8&node=21576558011

[7] AmbiQmicro 2019. *Apollo3 Blue Datasheet.* Retrieved January 11, 2022 from https://cdn.sparkfun.com/assets/learn_tutorials/9/0/9/Apollo3_Blue_MCU_Data_Sheet_v0_9_1.pdf

[8] Pedro Andrade, Ivanovitch Silva, Gabriel Signoretti, Marianne Silva, João Dias, Lucas Marques, and Daniel G Costa. 2021. An Unsupervised TinyML Approach Applied for Pavement Anomalies Detection Under the Internet of Intelligent Vehicles. In *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4. 0&IoT)*. IEEE, 642–647.

[9] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, 3 (2017), 1–18.

[10] Apache TVM 2020. *Apache TVM.* Retrieved January 11, 2022 from https://tvm.apache.org/

[11] Apple 2022. *HomePod mini.* Retrieved January 8, 2022 from https://www.apple.com/homepod-mini/

[12] Arducam 2021. *Arducam Pico4ML TinyML Dev Kit.* Retrieved December 30, 2021 from https://www.arducam.com/docs/pico/arducam-pico4mltinymldevkit/

[13] ArduCam 2022. *ArduCam 0.3MP: OV7675.* Retrieved January 8, 2022 from https://www.arducam.com/products/camera-breakout-board/0-3mp-ov7675/

[14] Arduino 2021. *Arduino Nano 33 BLE Sense.* Retrieved December 30, 2021 from https://store-usa.arduino.cc/products/arduino-nano-33-ble-sense

[15] Arduino 2021. *Portenta H7.* Retrieved December 30, 2021 from https://www.arduino.cc/pro/hardware/product/portenta-h7

[16] Arduino 2022. *Arduino.* Retrieved January 8, 2022 from https://www.arduino.cc/

[17] Arduino Project Hub 2019. *Cough Detection with TinyML on Arduino.* Retrieved December 19, 2021 from https://create.arduino.cc/projecthub/edge-impulse/cough-detection-with-tinyml-on-arduino-417f37

[18] Arduino Project Hub 2021. *TinyML-Language Detector-Based on Edge Impulse and Arduino © MIT.* Retrieved January 11, 2022 from https://create.arduino.cc/projecthub/enzo2/tinyml-language-detector-based-on-edge-impulse-arduino-f5cfa8?ref=part&ref_id=107215&offset=0

[19] Arduino Project Hub 2021. *TinyML: Speech Commands Detection.* Retrieved January 11, 2022 from https://create.arduino.cc/projecthub/ugotan/tinyml-speech-commands-detection-a3b51b?ref=part&ref_id=107215&offset=1

[20] Arm 2022. *Arm.* Retrieved January 8, 2022 from https://www.arm.com/

[21] Aruba Networks 2019. *Secure Wi-Fi For Healthcare Applications.* Retrieved October 23, 2021 from https://www.arubanetworks.com/assets/wp/WPHealthcareWLAN.pdf

[22] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. *arXiv preprint arXiv:1909.01377* (2019).

[23] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems* 3 (2021).

[24] Colby R Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, et al. 2020. Benchmarking TinyML systems: Challenges and direction. *arXiv preprint arXiv:2003.04821* (2020).

[25] Mandrita Banerjee, Junghee Lee, and Kim-Kwang Raymond Choo. 2018. A blockchain future for internet of things security: a position paper. *Digital Communications and Networks* 4, 3 (Aug. 2018), 149–160. https://doi.org/10.1016/j.dcan.2017.10.006

[26] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. 2018. Aciq: Analytical clipping for integer quantization of neural networks. (2018).

[27] Paul Palomero Bernardo, Christoph Gerum, Adrian Frischknecht, Konstantin Lübeck, and Oliver Bringmann. 2020. Ultratrail: A configurable ultralow-power tc-resnet ai accelerator for efficient keyword spotting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 4240–4251.

[28] Sizhen Bian and Paul Lukowicz. 2021. Capacitive Sensing Based On-board Hand Gesture Recognition with TinyML. In *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*. 4–5.

[29] Oliver Bringmann, Wolfgang Ecker, Ingo Feldner, Adrian Frischknecht, Christoph Gerum, Timo Hämäläinen, Muhammad Abdullah Hanif, Michael J Klaiber, Daniel Mueller-Gritschneder, Paul Palomero Bernardo, et al. 2021. Automated HW/SW Co-design for Edge AI: State, Challenges and Steps Ahead: Special Session Paper. In *2021 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 11–20.

[30] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[31] Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. 2021. Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus. *IEEE Trans. Comput.* (2021).

[32] Paul M Cabacungan, Carlos M Oppus, Nerissa G Cabacungan, John Paul A Mamaradlo, Paul Ryan A Santiago, Neil Angelo M Mercado, E Vincent S Faustino, and Gregory L Tangonan. 2021. Design and Development of A-vent: A Low-Cost Ventilator with Cost-Effective Mobile Cloud Caching and Embedded Machine Learning. In *2021 IEEE Region 10 Symposium (TENSYMP)*. IEEE, 1–8.

[33] Léopold Cambier, Anahita Bhiwandiwalla, Ting Gong, Mehran Nekuii, Oguz H Elibol, and Hanlin Tang. 2020. Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks. *arXiv preprint arXiv:2001.05674* (2020).
[34] Erick Cantú-Paz. 2003. Pruning neural networks with distribution estimation algorithms. In *Genetic and Evolutionary Computation Conference*. Springer, 790–800.
[35] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.
[36] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *International conference on machine learning*. PMLR, 2285–2294.
[37] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52, 1 (2016), 127–138.
[38] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308.
[39] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 27–39.
[40] Konstantinos Christidis and Michael Devetsikiotis. 2016. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access* 4 (2016), 2292–2303. https://doi.org/10.1109/ACCESS.2016.2566339
[41] Codelabs 2020. *AI Speech Recognition with TensorFlow Lite for Microcontrollers and SparkFun Edge*. Retrieved January 3, 2022 from https://codelabs.developers.google.com/codelabs/sparkfun-tensorflow/#0
[42] Coursera 2022. *Computer Vision with Embedded Machine Learning*. Retrieved January 3, 2022 from https://www.coursera.org/learn/computer-vision-with-embedded-machine-learning
[43] Coursera 2022. *Coursera*. Retrieved January 8, 2022 from https://www.coursera.org/
[44] Coursera 2022. *Introduction to Embedded Machine Learning*. Retrieved January 3, 2022 from https://www.coursera.org/learn/introduction-to-embedded-machine-learning
[45] Giulia Crocioni, Giambattista Gruosso, Danilo Pau, Davide Denaro, Luigi Zambrano, and Giuseppe Di Giore. 2021. Characterization of Neural Networks Automatically Mapped on Automotive-grade Microcontrollers. *arXiv preprint arXiv:2103.00201* (2021).
[46] Raj Dabre and Atsushi Fujita. 2019. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 6292–6299.
[47] Bin Dai, Chen Zhu, Baining Guo, and David Wipf. 2018. Compressing neural networks using the variational information bottleneck. In *International Conference on Machine Learning*. PMLR, 1135–1144.
[48] DaleGia 2020. *The Hacky Super Loop Arduino Nano 33 BLE Sense Example You Have Been Waiting For*. Retrieved January 11, 2022 from https://dalegi.com/2020/06/09/the-hacky-super-loop-arduino-nano-33-ble-sense-example-you-have-been-waiting-for/
[49] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. *Proceedings of Machine Learning and Systems* 3 (2021).
[50] Miguel de Prado, Manuele Rusci, Alessandro Capotondi, Romain Donze, Luca Benini, and Nuria Pazos. 2021. Robustifying the Deployment of tinyML Models for Autonomous mini-vehicles. *Sensors* 21, 4 (2021), 1339.
[51] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
[52] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. 2013. Predicting parameters in deep learning. *arXiv preprint arXiv:1306.0543* (2013).
[53] Tim Dettmers. 2015. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561* (2015).
[54] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[55] Hiroshi Doyu, Roberto Morabito, and Martina Brachmann. 2021. A tinymlaas ecosystem for machine learning in iot: Overview and research challenges. In *2021 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, 1–5.
[56] Bardienus P Duisterhof, Srivatsan Krishnan, Jonathan J Cruz, Colby R Banbury, William Fu, Aleksandra Faust, Guido CHE de Croon, and Vijay Janapa Reddi. 2019. Learning to seek: Autonomous source seeking with deep reinforcement learning onboard a nano drone microcontroller. *arXiv preprint arXiv:1909.11236* (2019).
[57] Abir Dutta and Shri Kant. 2021. Implementation of Cyber Threat Intelligence Platform on Internet of Things (IoT) using TinyML Approach for Deceiving Cyber Invasion. In *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE, 1–6.
[58] Lachit Dutta and Swapna Bharali. 2021. TinyML Meets IoT: A Comprehensive Survey. *Internet of Things* 16 (2021), 100461.

[59] Edge Impulse 2022. *Announcing Intro to Embedded Machine Learning on Coursera.* Retrieved January 8, 2022 from https://www.edgeimpulse.com/blog/announcing-intro-to-embedded-machine-learning-on-coursera
[60] Edge Impulse 2022. *Edge Impulse.* Retrieved January 8, 2022 from https://www.edgeimpulse.com/
[61] edX 2022. *edX.* Retrieved January 8, 2022 from https://www.edx.org
[62] edX 2022. *The Future of ML is Tiny and Bright.* Retrieved January 3, 2022 from https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning
[63] EEworldONLINE 2019. *Part 1: Microcontrollers and Microprocessors Continue Rapid Market Growth.* Retrieved January 8, 2022 from https://www.eeworldonline.com/part-1-microcontrollers-and-microprocessors-continue-rapid-market-growth/
[64] David Eigen, Jason Rolfe, Rob Fergus, and Yann LeCun. 2013. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847* (2013).
[65] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* 20, 1 (2019), 1997–2017.
[66] A.P. Engelbrecht. 2001. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Transactions on Neural Networks* 12, 6 (2001), 1386–1399. https://doi.org/10.1109/72.963775
[67] Espressif 2021. *ESP32-DevKitC.* Retrieved December 30, 2021 from https://www.espressif.com/en/products/devkits/esp32-devkitc/overview
[68] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. 2021. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. *arXiv preprint arXiv:2103.05579* (2021).
[69] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320* (2020).
[70] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul N Whatmough. 2019. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. *arXiv preprint arXiv:1905.12107* (2019).
[71] Igor Fedorov, Marko Stamenovic, Carl Jensen, Li-Chia Yang, Ari Mandell, Yiming Gan, Matthew Mattina, and Paul N Whatmough. 2020. TinyLSTMs: Efficient neural speech enhancement for hearing aids. *arXiv preprint arXiv:2005.11138* (2020).
[72] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca Benini. 2018. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 1–4.
[73] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).
[74] Daichi Fujiki, Xiaowei Wang, Arun Subramaniyan, and Reetuparna Das. 2021. In-/Near-Memory Computing. *Synthesis Lectures on Computer Architecture* 16, 2 (2021), 1–140.
[75] Sukhpal Singh Gill, Minxian Xu, Carlo Ottaviani, Panos Patros, Rami Bahsoon, Arash Shaghaghi, Muhammed Golec, Vlado Stankovski, Huaming Wu, Ajith Abraham, Manmeet Singh, Harshit Mehta, Soumya K. Ghosh, Thar Baker, Ajith Kumar Parlikad, Hanan Lutfiyya, Salil S. Kanhere, Rizos Sakellariou, Schahram Dustdar, Omer Rana, Ivona Brandic, and Steve Uhlig. 2022. AI for next generation computing: Emerging trends and future directions. *Internet of Things* 19 (Aug. 2022), 100514. https://doi.org/10.1016/j.iot.2022.100514
[76] Marco Giordano, Philipp Mayer, and Michele Magno. 2020. A Battery-Free Long-Range Wireless Smart Camera for Face Detection. In *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. 29–35.
[77] GitHub 2020. *TinyML Example: Anomaly Detection.* Retrieved January 3, 2022 from https://github.com/ShawnHymel/tinyml-example-anomaly-detection
[78] GitHub 2021. *CurrentSense-TinyML.* Retrieved January 3, 2022 from https://github.com/Santandersecurityresearch/CurrentSense-TinyML
[79] GitHub 2021. *Embedded Learning Library.* Retrieved January 8, 2022 from https://github.com/microsoft/ELL
[80] GitHub 2021. *TinyML Study Group.* Retrieved January 3, 2022 from https://github.com/scaledown-team/study-group
[81] GitHub 2021. *uTensor.* Retrieved January 8, 2022 from https://github.com/uTensor/uTensor
[82] Google 2022. *Google.* Retrieved January 8, 2022 from https://www.google.com/
[83] Google 2022. *Google Nest.* Retrieved January 8, 2022 from https://support.google.com/googlenest/answer/7130274?hl=en
[84] Qiushan Guo, Zhipeng Yu, Yichao Wu, Ding Liang, Haoyu Qin, and Junjie Yan. 2019. Dynamic recursive neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5147–5156.
[85] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International conference on machine learning*. PMLR, 1737–1746.
[86] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. 2018. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE transactions on neural networks and learning systems* 29, 11 (2018), 5784–5789.
[87] Hackster.io 2020. *Easy TinyML on ESP32 and Arduino.* Retrieved January 3, 2022 from https://www.hackster.io/news/easy-tinyml-on-esp32-and-arduino-a9dbc509f26c

[88] Hackster.io 2020. *Handwriting Recognition.* Retrieved January 3, 2022 from https://www.hackster.io/naveenbskumar/handwriting-recognition-7583e3
[89] Hackster.io 2021. *TapLock - A bike lock with machine learning.* Retrieved January 3, 2022 from https://www.hackster.io/taplock/taplock-a-bike-lock-with-machine-learning-85641c
[90] Masafumi Hagiwara. 1993. Removal of hidden units and weights for back propagation networks. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, Vol. 1. IEEE, 351–354.
[91] Rahman Hajian, S ZakeriKia, SH Erfani, and M Mirabi. 2020. SHAPARAK: Scalable healthcare authentication protocol with attack-resilience and anonymous key-agreement. *Computer Networks* 183 (2020), 107567.
[92] Hong-Gui Han and Jun-Fei Qiao. 2013. A structure optimisation algorithm for feedforward neural network construction. *Neurocomputing* 99 (2013), 347–357.
[93] Harvard 2022. *Harvard University.* Retrieved January 8, 2022 from https://www.harvard.edu/
[94] Babak Hassibi, David G Stork, and Gregory J Wolff. 1993. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*. IEEE, 293–299.
[95] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*. 784–800.
[96] Lennart Heim, Andreas Biri, Zhongnan Qu, and Lothar Thiele. 2021. Measuring what Really Matters: Optimizing Neural Networks for TinyML. *arXiv preprint arXiv:2104.10645* (2021).
[97] Himax 2021. *Himax WE-I.* Retrieved December 30, 2021 from https://www.himax.com.tw/products/intelligent-sensing/always-on-smart-sensing/
[98] Lu Hou, Quanming Yao, and James T Kwok. 2016. Loss-aware binarization of deep networks. *arXiv preprint arXiv:1611.01600* (2016).
[99] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7132–7141.
[100] Yiming Hu, Siyang Sun, Jianquan Li, Xingang Wang, and Qingyi Gu. 2018. A novel channel pruning method for deep neural network compression. *arXiv preprint arXiv:1805.11394* (2018).
[101] IC Insights 2020. *MCUs Expected to Make Modest Comeback After 2020 Drop.* Retrieved January 8, 2022 from https://www.icinsights.com/news/bulletins/MCUs-Expected-To-Make-Modest-Comeback-After-2020-Drop--/
[102] Gian Marco Iodice. 2022. *TinyML Cookbook combine artificial intelligence and ultra-low-power embedded devices to make... the world smarter.* Packt Publishing Limited, S.l. OCLC: 1309923814.
[103] Rahul Jain, Vijay Bhaskar Semwal, and Praveen Kaushik. 2021. Deep ensemble learning approach for lower extremity activities recognition using wearable sensors. *Expert Systems* (2021), e12743.
[104] Nikhil Jangamreddy. 2019. A Survey on Specialised Hardware for Machine Learning. (2019).
[105] Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. 2020. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 4805–4815.
[106] Raj Kamal, Aabha Jain, and Manojkumar Vilasrao Deshpande. 2022. Architectural Design for Inspection of Machine Objects Using Small DNNs as TinyML for Machine Vision of Defects and Faults in the Manufacturing Processes. In *Smart Systems: Innovations in Computing*. Springer, 377–387.
[107] Ehud D Karnin. 1990. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks* 1, 2 (1990), 239–242.
[108] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. 2019. Edge computing: A survey. *Future Generation Computer Systems* 97 (2019), 219–235.
[109] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Deeply-recursive convolutional network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1637–1645.
[110] Okan Köpüklü, Maryam Babaee, Stefan Hörmann, and Gerhard Rigoll. 2019. Convolutional neural networks with layer reuse. In *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 345–349.
[111] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).
[112] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.
[113] Nir Kshetri. 2017. Blockchain's roles in strengthening cybersecurity and protecting privacy. *Telecommunications Policy* 41, 10 (Nov. 2017), 1027–1038. https://doi.org/10.1016/j.telpol.2017.09.003
[114] Jisu Kwon and Daejin Park. 2021. Hardware/Software Co-Design for TinyML Voice-Recognition Application on Resource Frugal Edge Devices. *Applied Sciences* 11, 22 (2021), 11073.
[115] Shashikant Vitthalrao Lahade, Srikanth Namuduri, Himanshu Upadhyay, and Shekhar Bhansali. 2020. Alcohol Sensor Calibration on the Edge Using Tiny Machine Learning (Tiny-ML) Hardware. In *ECS Meeting Abstracts*. IOP Publishing, 1848.

[116] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601* (2018).
[117] Maximilian Lam, Sharad Chitlangia, Srivatsan Krishnan, Zishen Wan, Gabriel Barth-Maron, Aleksandra Faust, and Vijay Janapa Reddi. 2019. Quantized reinforcement learning (quarl). *arXiv preprint arXiv:1910.01055* (2019).
[118] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).
[119] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*. 598–605.
[120] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* (2018).
[121] Davide Nadalini Alessandro Capotondi Francesco Conti Leonardo Ravaglia, Manuele Rusci and Luca Benini. 2021. A TinyML Platform for On-Device Continual Learning With Quantized Latent Replays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2021).
[122] Jiajun Li, Guihai Yan, Wenyan Lu, Shuhao Jiang, Shijun Gong, Jingya Wu, and Xiaowei Li. 2018. SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 343–348.
[123] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. 2018. Accelerating Convolutional Networks via Global & Dynamic Filter Pruning.. In *IJCAI*, Vol. 2. 8.
[124] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. 2019. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2790–2799.
[125] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[126] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[127] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).
[128] Mansoureh Lord. 2021. *TinyML, Anomaly Detection*. Ph.D. Dissertation. California State University, Northridge.
[129] Christos Louizos, Karen Ullrich, and Max Welling. 2017. Bayesian compression for deep learning. *arXiv preprint arXiv:1705.08665* (2017).
[130] Tom H Luan, Longxiang Gao, Zhi Li, Yang Xiang, Guiyi Wei, and Limin Sun. 2015. Fog computing: Focusing on mobile users at the edge. *arXiv preprint arXiv:1502.01815* (2015).
[131] Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. 2022. Online continual learning in image classification: An empirical survey. *Neurocomputing* 469 (2022), 28–51.
[132] Arun Mallya and Svetlana Lazebnik. 2018. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *arXiv preprint arXiv:1801.06519* 6, 8 (2018).
[133] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. 2017. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2322–2358.
[134] Muhammad Ali Mazidi, Janice Gillispie Mazidi, and Rolin D McKinlay. 2006. *The 8051 microcontroller and embedded systems: using Assembly and C*. Vol. 626. Pearson/Prentice Hall.
[135] John T McCoy and Lidia Auret. 2019. Machine learning applications in minerals processing: A review. *Minerals Engineering* 132 (2019), 95–109.
[136] Marci Meingast, Tanya Roosta, and Shankar Sastry. 2006. Security and privacy issues with health care information technology. In *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 5453–5458.
[137] Tal Melamed. 2018. An active man-in-the-middle attack on bluetooth smart devices. *Safety and Security Studies (2018)* 15 (2018), 2018.
[138] Naveen Mellempudi, Sudarshan Srinivasan, Dipankar Das, and Bharat Kaul. 2019. Mixed precision training with 8-bit floating point. *arXiv preprint arXiv:1905.12334* (2019).
[139] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K Esser, and Dharmendra Modha. 2016. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981* (2016).
[140] Szymon Migacz. 2017. 8-bit inference with tensorrt. In *GPU technology conference*, Vol. 2. 5.
[141] MLCommons 2021. *MLCommons aims to accelerate machine learning innovation to benefit everyone.* Retrieved January 11, 2022 from https://www.tensorflow.org/lite
[142] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. 2017. Variational Dropout Sparsifies Deep Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 2498–2507. https://proceedings.mlr.press/v70/molchanov17a.html

[143] Kathryn Montgomery, Jeff Chester, and Katharina Kopp. 2018. Health wearables: ensuring fairness, preventing discrimination, and promoting equity in an emerging Internet-of-Things environment. *Journal of Information Policy* 8 (2018), 34–77.
[144] Michael C Mozer and Paul Smolensky. 1989. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*. 107–115.
[145] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2020. A modern primer on processing in memory. *arXiv preprint arXiv:2012.03112* (2020).
[146] Pramod L Narasimha, Walter H Delashmit, Michael T Manry, Jiang Li, and Francisco Maldonado. 2008. An integrated growing-pruning method for feedforward network training. *Neurocomputing* 71, 13-15 (2008), 2831–2847.
[147] James O' Neill. 2020. An overview of neural network compression. *arXiv preprint arXiv:2006.03669* (2020).
[148] Justin Nguyen, Reese Grimsley, and Bob Iannucci. 2021. TrafficNNode: Low Power Vehicle Sensing Platform for Smart Cities. In *2021 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 278–282.
[149] Steven J Nowlan and Geoffrey E Hinton. 1992. Simplifying neural networks by soft weight-sharing. *Neural computation* 4, 4 (1992), 473–493.
[150] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. 2020. Asap: Architecture search, anneal and prune. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 493–503.
[151] NVIDIA 2021. *Jetson Nano Developer Kit.* Retrieved December 30, 2021 from https://developer.nvidia.com/embedded/jetson-nano-developer-kit
[152] NXP 2022. *eIQ® ML Software Development Environment.* Retrieved January 11, 2022 from https://www.nxp.com/design/software/development-software/eiq-ml-development-environment:EIQ
[153] Samson Otieno Ooko, Didacienne Mukanyiligira, Jean Pierre Munyampundu, and Jimmy Nsenga. 2021. Synthetic Exhaled Breath Data-Based Edge AI Model for the Prediction of Chronic Obstructive Pulmonary Disease. In *2021 International Conference on Computing and Communications Applications and Technologies (I3CAT)*. IEEE, 1–6.
[154] Charalampos Orfanidis, Rayén Bel Haj Hassen, Armando Kwiek, Xenofon Fafoutis, and Martin Jacobsson. 2021. A Discreet Wearable Long-Range Emergency System Based on Embedded Machine Learning. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 182–187.
[155] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. 2021. PhiNets: a scalable backbone for low-power AI at the edge. *arXiv preprint arXiv:2110.00337* (2021).
[156] Zhengyuan Pang, Lifeng Sun, Zhi Wang, Erfang Tian, and Shiqiang Yang. 2015. A survey of cloudlet based mobile computing. In *2015 International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 268–275.
[157] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. 2018. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 580–595.
[158] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *International Conference on Machine Learning*. PMLR, 4055–4064.
[159] Aditya Jyoti Paul, Puranjay Mohan, and Stuti Sehgal. 2020. Rethinking generalization in american sign language prediction for edge devices with extremely low memory footprint. In *2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*. IEEE, 147–152.
[160] Hongwu Peng, Shaoyi Huang, Tong Geng, Ang Li, Weiwen Jiang, Hang Liu, Shusen Wang, and Caiwen Ding. 2021. Accelerating Transformer-based Deep Learning Models on FPGAs using Column Balanced Block Pruning. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 142–148.
[161] Andrew Perrin. 2019. Digital gap between rural and nonrural America persists. *Pew Research Center* (2019).
[162] Bryan A Plummer, Nikoli Dryden, Julius Frost, Torsten Hoefler, and Kate Saenko. 2020. Shapeshifter networks: Cross-layer parameter sharing for scalable and effective deep learning. *arXiv e-prints* (2020), arXiv–2006.
[163] Panjie Qi, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Hongwu Peng, Shaoyi Huang, Zhenglun Kong, Yuhong Song, and Bingbing Li. 2021. Accelerating Framework of Transformer by Hardware Design and Model Compression Co-Optimization. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.
[164] Sajay Rai, Philip Chukwuma, and Richard Cozart. 2016. *Security and Auditing of Smart Devices: Managing Proliferation of Confidential Data on Corporate and BYOD Devices.* Auerbach Publications.
[165] Raspberry Pi 2021. *Raspberry Pi.* Retrieved January 8, 2022 from https://www.raspberrypi.org/
[166] Raspberry Pi 2021. *Raspberry Pi 4.* Retrieved December 30, 2021 from https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[167] Raspberry Pi 2021. *Raspberry Pi Pico.* Retrieved December 30, 2021 from https://www.raspberrypi.com/products/raspberry-pi-pico/
[168] Leonardo Ravaglia, Manuele Rusci, Davide Nadalini, Alessandro Capotondi, Francesco Conti, and Luca Benini. 2021. A TinyML Platform for On-Device Continual Learning With Quantized Latent Replays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11, 4 (2021), 789–802.
[169] Wamiq Raza, Anas Osman, Francesco Ferrini, and Francesco De Natale. 2021. Energy-Efficient Inference on the Edge Exploiting TinyML Capabilities for UAVs. *Drones* 5, 4 (2021), 127.

[170] Vijay Janapa Reddi, Brian Plancher, Susan Kennedy, Laurence Moroney, Pete Warden, Anant Agarwal, Colby Banbury, Massimo Banzi, Matthew Bennett, Benjamin Brown, et al. 2021. Widening Access to Applied Machine Learning with TinyML. *arXiv preprint arXiv:2106.04008* (2021).
[171] Russell Reed. 1993. Pruning algorithms-a survey. *IEEE transactions on Neural Networks* 4, 5 (1993), 740–747.
[172] Haoyu Ren, Darko Anicic, and Thomas Runkler. 2021. TinyOL: TinyML with Online-Learning on Microcontrollers. *arXiv preprint arXiv:2103.08295* (2021).
[173] Francesco Restuccia and Tommaso Melodia. 2020. Deepwierl: Bringing deep reinforcement learning to the internet of self-adaptive things. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 844–853.
[174] A Navaas Roshan, B Gokulapriyan, C Siddarth, and Priyanka Kokil. 2021. Adaptive Traffic Control With TinyML. In *2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE, 451–455.
[175] Firdose Saeik, Marios Avgeris, Dimitrios Spatharakis, Nina Santi, Dimitrios Dechouniotis, John Violos, Aris Leivadeas, Nikolaos Athanasopoulos, Nathalie Mitton, and Symeon Papavassiliou. 2021. Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks* 195 (Aug. 2021), 108177. https://doi.org/10.1016/j.comnet.2021.108177
[176] Ramon Sanchez-Iborra and Antonio F Skarmeta. 2020. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine* 20, 3 (2020), 4–18.
[177] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
[178] Pedro Savarese and Michael Maire. 2019. Learning implicitly recurrent CNNs through parameter sharing. *arXiv preprint arXiv:1902.09701* (2019).
[179] Scaledown 2022. *Scaledown.* Retrieved January 11, 2022 from https://scaledown-team.github.io/
[180] Seedstudio 2021. *Everything About TinyML – Basics, Courses, Projects & More!* Retrieved January 3, 2022 from https://www.seeedstudio.com/blog/2021/06/14/everything-about-tinyml-basics-courses-projects-more/
[181] Seedstudio 2021. *Wio Terminal - Hello World of AI, an Azure certified device to get started with IoT and TinyML.* Retrieved January 3, 2022 from https://www.seeedstudio.com/wio-terminal-tinyml.html
[182] Seeedstudio 2021. *Seeeduino-XIAO.* Retrieved December 30, 2021 from https://www.seeedstudio.com/Seeeduino-XIAO-Arduino-Microcontroller-SAMD21-Cortex-M0+-p-4426.html
[183] Seeedstudio 2021. *Wio Terminal.* Retrieved December 30, 2021 from https://www.seeedstudio.com/Wio-Terminal-p-4509.html
[184] Suranga Seneviratne, Yining Hu, Tham Nguyen, Guohao Lan, Sara Khalifa, Kanchana Thilakarathna, Mahbub Hassan, and Aruna Seneviratne. 2017. A survey of wearable devices and challenges. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2573–2620.
[185] Rudy Setiono and Wee Kheng Leow. 2000. Pruned neural networks for regression. In *Pacific Rim International Conference on Artificial Intelligence*. Springer, 500–509.
[186] Muhammad Shafique, Theocharis Theocharides, Vijay Janapa Reddy, and Boris Murmann. 2021. TinyML: Current Progress, Research Challenges, and Future Roadmap. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1303–1306.
[187] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. 2009. Hash kernels for structured data. *Journal of Machine Learning Research* 10, 11 (2009).
[188] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.
[189] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).
[190] Ajay Shrestha and Ausif Mahmood. 2019. Review of Deep Learning Algorithms and Architectures. *IEEE Access* 7 (2019), 53040–53065. https://doi.org/10.1109/ACCESS.2019.2912200
[191] Shachar Siboni, Asaf Shabtai, Nils O Tippenhauer, Jemin Lee, and Yuval Elovici. 2016. Advanced security testbed framework for wearable IoT devices. *ACM Transactions on Internet Technology (TOIT)* 16, 4 (2016), 1–25.
[192] Yuhong Song, Weiwen Jiang, Bingbing Li, Panjie Qi, Qingfeng Zhuge, Edwin Hsing-Mean Sha, Sakyasingha Dasgupta, Yiyu Shi, and Caiwen Ding. 2021. Dancing along Battery: Enabling Transformer with Run-time Reconfigurability on Mobile Devices. *arXiv preprint arXiv:2102.06336* (2021).
[193] Sony 2021. *Spresense.* Retrieved December 30, 2021 from https://developer.sony.com/develop/spresense/
[194] Stanislava Soro. 2021. TinyML for Ubiquitous Edge AI. *arXiv preprint arXiv:2102.01255* (2021).
[195] SparkFun 2021. *SparkFun Edge Development Board - Apollo3 Blue.* Retrieved December 30, 2021 from https://www.sparkfun.com/products/15170
[196] STM32L4 Discovery kit 2021. *STM32L4 Discovery kit.* Retrieved December 30, 2021 from https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html#overview&secondary=st_all-features_sec-nav-tab
[197] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. 2019. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686* (2019).

[198] Filip Svoboda, David Nunes, Milad Alizadeh, Russel Daries, Rui Luo, Akhil Mathur, Sourav Bhattacharya, Jorge Sa Silva, and Nicholas Donald Lane. 2020. Resource Efficient Deep Reinforcement Learning for Acutely Constrained TinyML Devices. In *Research Symposium on Tiny Machine Learning*.

[199] SYNTIANT 2019. *The Speed and Power Advantage of a Purpose-Built Neural Compute Engine.* Retrieved January 11, 2022 from https://www.syntiant.com/post/keyword-spotting-power-comparison

[200] Syntiant 2021. *Syntiant Tiny Machine Learning Development Board.* Retrieved December 30, 2021 from https://www.syntiant.com/tinyml

[201] Enrico Tabanelli, Giuseppe Tagliavini, and Luca Benini. 2021. DNN is not all you need: Parallelizing Non-Neural ML Algorithms on Ultra-Low-Power IoT Processors. *arXiv preprint arXiv:2107.09448* (2021).

[202] Ying Tai, Jian Yang, and Xiaoming Liu. 2017. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3147–3155.

[203] Paul J. Taylor, Tooska Dargahi, Ali Dehghantanha, Reza M. Parizi, and Kim-Kwang Raymond Choo. 2020. A systematic literature review of blockchain cyber security. *Digital Communications and Networks* 6, 2 (May 2020), 147–156. https://doi.org/10.1016/j.dcan.2019.01.005

[204] TensorFlow Blog 2021. *Building a TinyML Application with TF Micro and SensiML.* Retrieved January 3, 2022 from https://blog.tensorflow.org/2021/05/building-tinyml-application-with-tf-micro-and-sensiml.html

[205] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. 2018. Faster gaze prediction with dense networks and Fisher pruning. *CoRR* abs/1801.05787 (2018). arXiv:1801.05787 http://arxiv.org/abs/1801.05787

[206] TinyML 2022. *TinyML Foundation.* Retrieved January 8, 2022 from https://www.tinyml.org/

[207] TinyML Book 2019. *TinyML Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers.* Retrieved January 3, 2022 from https://tinymlbook.com/

[208] Kristof T'Jonck, Chandrakanth R Kancharla, Jens Vankeirsbilck, Hans Hallez, Jeroen Boydens, and Bozheng Pang. 2021. Real-Time Activity Tracking using TinyML to Support Elderly Care. In *2021 XXX International Scientific Conference Electronics (ET)*. IEEE, 1–6.

[209] Vasileios Tsoukas, Anargyros Gkogkidis, Aikaterini Kampa, Georgios Spathoulas, and Athanasios Kakarountas. 2022. Enhancing Food Supply Chain Security through the Use of Blockchain and TinyML. *Information* 13, 5 (May 2022), 213. https://doi.org/10.3390/info13050213 Number: 5 Publisher: Multidisciplinary Digital Publishing Institute.

[210] Juanjuan Tu, Yongzhao Zhan, and Fei Han. 2010. A neural network pruning method optimized with PSO algorithm. In *2010 Second International Conference on Computer Modeling and Simulation*, Vol. 3. IEEE, 257–259.

[211] Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008* (2017).

[212] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[213] Federico Venturini, Federico Mason, Francesco Pase, Federico Chiariotti, Alberto Testolin, Andrea Zanella, and Michele Zorzi. 2020. Distributed reinforcement learning for flexible UAV swarm control with transfer learning capabilities. In *Proceedings of the 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*. 1–6.

[214] Ishan Karunanayake Visal Rajapakse and Nadeem Ahmed. 2023. Intelligence at the Extreme Edge: A Survey on Reformable TinyML. *ACM Comput. Surv. 55, 13s* (2023).

[215] Miljan Vuletic, Vladimir Mujagic, Nikola Milojevic, and Debmalya Biswas. [n.d.]. Edge AI Framework for Healthcare Applications. ([n.d.]).

[216] Chandrasekar Vuppalapati, Anitha Ilapakurti, Karthik Chillara, Sharat Kedari, and Vanaja Mamidi. 2020. Automating Tiny ML Intelligent Sensors DevOPS Using Microsoft Azure. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2375–2384.

[217] Chandrasekar Vuppalapati, Anitha Ilapakurti, Sharat Kedari, Jaya Vuppalapati, Santosh Kedari, and Raja Vuppalapati. 2020. Democratization of AI, Albeit Constrained IoT Devices & Tiny ML, for Creating a Sustainable Food Future. In *2020 3rd International Conference on Information and Computer Technologies (ICICT)*. IEEE, 525–530.

[218] Chandrasekar Vuppalapati, Anitha Ilapakurti, Sharat Kedari, Raja Vuppalapati, Jaya Vuppalapati, and Santosh Kedari. 2021. Crossing the Artificial Intelligence (AI) Chasm, Albeit Using Constrained IoT Edges and Tiny ML, for Creating a Sustainable Food Future. In *Proceedings of Fifth International Congress on Information and Communication Technology*. Springer, 540–553.

[219] Christopher A Walsh. 2013. Peter Huttenlocher (1931–2013). *Nature* 502, 7470 (2013), 172–172.

[220] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8612–8620.

[221] Shengling Wang, Rongfang Bie, Feng Zhao, Nan Zhang, Xiuzhen Cheng, and Hyeong-Ah Choi. 2016. Security in wearable communications. *IEEE Network* 30, 5 (2016), 61–67.

[222] Pete Warden and Daniel Situnayake. 2019. *TinyML.* O'Reilly Media, Incorporated.

[223] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*. 1113–1120.

[224] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems* 29 (2016), 2074–2082.

[225] Darrell Whitley, Timothy Starkweather, and Christopher Bogart. 1990. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing* 14, 3 (1990), 347–361.
[226] Alexander Wong, Mahmoud Famouri, Maya Pavlova, and Siddharth Surana. 2020. Tinyspeech: Attention condensers for deep speech recognition neural networks on edge devices. *arXiv preprint arXiv:2008.04245* (2020).
[227] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. 2019. Sharing attention weights for fast transformer. *arXiv preprint arXiv:1906.11024* (2019).
[228] Lei Yang, Weiwen Jiang, Weichen Liu, HM Edwin, Yiyu Shi, and Jingtong Hu. 2020. Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 85–90.
[229] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).
[230] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. 2017. A survey on the edge computing for the Internet of Things. *IEEE access* 6 (2017), 6900–6919.
[231] Dejiao Zhang, Haozhu Wang, Mario Figueiredo, and Laura Balzano. 2018. Learning to share: Simultaneous parameter tying and sparsification in deep learning. In *International Conference on Learning Representations*.
[232] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128* (2017).
[233] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. 2018. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2472–2481.
[234] Andy Zhou, Rikky Muller, and Jan Rabaey. 2021. Memory-Efficient, Limb Position-Aware Hand Gesture Recognition using Hyperdimensional Computing. *arXiv preprint arXiv:2103.05267* (2021).
[235] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. 2018. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 9426–9435.
[236] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).
[237] Xingyu Zhou, Zhuangwei Kang, Robert Canady, Shunxing Bao, Daniel Allen Balasubramanian, and Aniruddha Gokhale. 2021. Exploring Cloud Assisted Tiny Machine Learning Application Patterns for PHM Scenarios. In *Annual Conference of the PHM Society*, Vol. 13.

Received 28 June 2022; revised 3 September 2023; accepted 11 April 2024