future internet

MDPI

Review

TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review

Nikolaos Schizas, Aristeidis Karras *, Christos Karras and Spyros Sioutas

Computer Engineering and Informatics Department, University of Patras, 26504 Patras, Greece
* Correspondence: akarras@ceid.upatras.gr

Abstract: The rapid emergence of low-power embedded devices and modern machine learning (ML) algorithms has created a new Internet of Things (IoT) era where lightweight ML frameworks such as TinyML have created new opportunities for ML algorithms running within edge devices. In particular, the TinyML framework in such devices aims to deliver reduced latency, efficient bandwidth consumption, improved data security, increased privacy, lower costs and overall network cost reduction in cloud environments. Its ability to enable IoT devices to work effectively without constant connectivity to cloud services, while nevertheless providing accurate ML services, offers a viable alternative for IoT applications seeking cost-effective solutions. TinyML intends to deliver on-premises analytics that bring significant value to IoT services, particularly in environments with limited connection. This review article defines TinyML, presents an overview of its benefits and uses and provides background information based on up-to-date literature. Then, we demonstrate the TensorFlow Lite framework which supports TinyML along with analytical steps for an ML model creation. In addition, we explore the integration of TinyML with network technologies such as 5G and LPWAN. Ultimately, we anticipate that this analysis will serve as an informational pillar for the IoT/Cloud research community and pave the way for future studies.

Keywords: Internet of Things; large scale IoT deployments; TinyML; edge computing; edge AI; edge ML; machine learning; deep learning; Tensorflow Lite; TFLM; LPWAN; 5G

Citation: Schizas, N.; Karras, A.; Karras, C.; Sioutas, S. TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review. *Future Internet* **2022**, *14*, 363. https://doi.org/10.3390/fi14120363

Academic Editor: Giovanni Pau

Received: 17 October 2022
Accepted: 30 November 2022
Published: 6 December 2022

1. Introduction

Data production has expanded at an extraordinary rate during the past decades [1], owing to the widespread deployment of sensor technology and the increasing digitization of the world. This tendency has been assisted by the increased processing and data storage capabilities offered by cloud technology. The growth in computational capabilities has expedited the study and development of deep neural networks (CNNs) [2,3], which have grown in complexity and resource demands through time. Cloud computing currently handles the most complex AI models with millions of attributes that require terabytes of memory and lightning-fast computation. Because they have infinite computing resources and memory, these massive neural network models are primarily concerned with accuracy and speed.

Mobile computing arose with the arrival of mobile devices such as tablets, smartphones and laptops. Mobile computing [4] is mostly concerned with wireless communications for mobile devices, which have profited from technological advancements such as low-power CPUs, tiny memory, and low-power display technologies. Shortly afterwards, mobile computing systems began to be applied in a variety of fields such as robots, autonomous vehicles, and augmented reality, which demand the performance of real-time tasks in a very short period without relying on cloud-based computing. This transition to mobile platforms has sparked the creation of a new type of neural networks [5,6] that are smaller in size and prioritize model efficiency above accuracy. Figure 1 depicts the advancement of such technologies (Cloud ML, Mobile ML and TinyML) over time from 2006 to 2019.

Figure 1. The advancement of computing technology over time.

Along with the advent of the artificial intelligence and machine learning sectors, advancements in connectivity and the proliferation of system-on-chip solutions have fueled the expansion of the Internet of Things (IoT) industry. Nowadays, the number of products incorporating microcontrollers (MCUs) is expanding at an exponential rate. Over 250 billion microcontrollers are in use, with 40 billion microcontrollers scheduled to be deployed by the end of 2022 [7]. This trend is further shown in Figure 2.

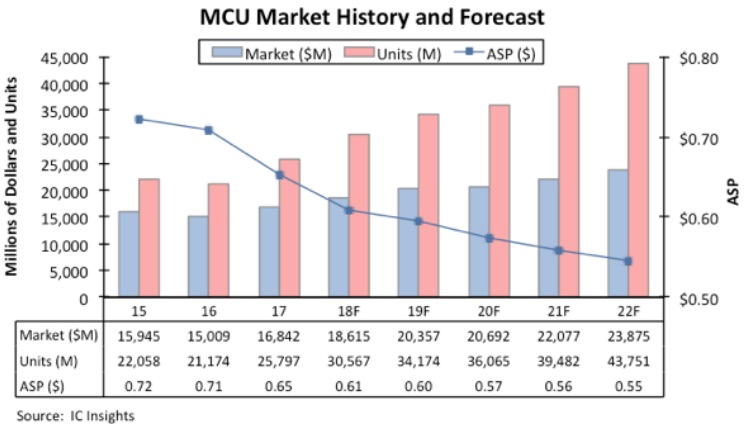

Figure 2. The history of the MCU market [7].

In several circumstances, however, embedded systems do not handle the data they gather; rather, the data are transported to a distant site for storage and subsequent processing. In some applications, this might result in undesirable delay, data leaks, and privacy concerns. This has sparked research into the fundamental difficulties associated with developing machine learning algorithms on embedded devices with restricted resources (in terms of memory, processor speed and power) [8].

Information is frequently gathered by embedded devices from a number of sensors (audio, IR cameras, IMU, accelerometers, temperature sensors, CMOS cameras, chemical sensors, etc.). These embedded systems often are powered by batteries and consume little energy (1 mW or less). Because that primarily performs simple processing tasks, the CPU of the device is often left idle over time. They are ideal for developing lightweight inference algorithms due to the ubiquitous usage of MCU-powered embedded devices, their latent processing capabilities, and their closeness to physical data sources via sensors. Tiny machine learning (TinyML), a new embedded technology industry aiming at building machine learning algorithms on resource-constrained devices based on MCUs, arose from this technique [9].

TinyML will allow advancements in a variety of fields, including consumer electronics, autonomous systems, healthcare, distributed cyber-physical systems, smart agriculture,

and the larger subject of artificial intelligence. Furthermore, accomplishing these objectives necessitates collaborative solutions from a variety of disciplines, including machine learning, optimization, hardware design, computer architecture and signal processing, among several others. In order to establish goals and address the difficulties of this novel and experimental, interdisciplinary topic, TinyML development requires a collaborative effort between embedded systems and the machine learning communities. We evaluate the present state of TinyML in this survey, which aims to provide artificial intelligence to smaller, less powerful edge devices, examine the TensorFlow Lite framework, which is among the most important frameworks for the growth of TinyML systems, and discuss a few enabling factors for future expansion of TinyML.

The absence of a suitable framework has delayed the acceptance and deployment of TinyML in applications. In addition to building a model on an embedded target, the framework should enable model training on a high computing platform. Model orchestration and testing, which are helpful for manufacturing devices, should be used by TinyML together with a broad ecosystem of machine learning technologies. In this paper, TFLM will be examined at in order to introduce deep learning to embedded devices and so significantly widen the applicability of ML. The TFLM is a framework designed particularly for deep learning on microcontrollers with extremely little storage (a few kilobytes). The key contributions of TFLM are its design decisions to solve the particular issues of embedded systems: Hardware diversity in a dispersed environment, missing software functionalities, and extreme resource restrictions.

In conclusion, this study presents the historical evolution of TinyML, a thorough description of this technology, and a methodical assessment of prior research efforts to offer specific recommendations for future research. In order to fill this gap, we highlight benefits of TinyML, applications, and use cases as well as the substantial contributions made by industry and academics. These contributions help scholars gain a deeper comprehension of the fundamental ideas of TinyML. The key contributions of this survey are as follows:

- Presentation of major performance measures for the TinyML framework, as well as its definition and overview; examination of important technologies.
- Review of research on TinyML conducted by various research groups and creation of an academic map.
- Identify important obstacles and give a future direction for TinyML research in which we cover numerous concerns.
- To assemble a list of existing TinyML-based toolkits for training and model construction at the edge, where hardware platforms, software programs, and libraries are available.
- Present and analyze TinyML application areas, as well as illustrate various TinyML use cases.

The remainder of the article is arranged as follows. A comprehensive overview of TinyML, with all of its features, is provided in Section 2. In Section 3, a detailed analysis of the TensorFlow Lite-TFLM framework for use in TinyML applications is provided. Section 4 illustrates the integration of TinyML with network technologies. Section 5 discusses the technology of TinyML and conclusions about it. Finally, Section 6 presents the future directions for TinyML that will have an important role in its development.

In continuation of the above, the goal of this journal is to provide a thorough overview of this ground-breaking technology to everyone with an interest in the topic, in addition to the in-depth research section. In conclusion, this systematic review will serve as an informative cornerstone for the research community, pave the way for further research in this direction, and serve as a roadmap for understanding the new, emerging field of TinyML.

2. TinyML

2.1. Overview

Because it has the potential to improve privacy, autonomy, responsiveness, and energy efficiency of edge devices, machine learning (ML)-based inference is becoming more and

more alluring [10–12]. Up until now, machine learning models like quantum, using pruning, and sparsity have significantly improved as a result of the majority of edge ML research focusing on inference on mobile devices. Meanwhile, substantial strides have been made in recent years to broaden the reach of edge systems. Expanding the use of edge ML to devices in the microcontroller category is of interest to both academia [10,11] and businesses [13,14].

The field of machine learning known as tiny machine learning (TinyML) is expanding quickly. It consists of hardware, software, and algorithms that can analyze sensor data using ultra-low power devices, opening up a wide range of continuously available use cases for battery-powered devices. TinyML systems are gradually being utilized for a variety of commercial applications, with additional systems on the horizon, as substantial breakthroughs in algorithms, networks, and models are produced. In addition, what were originally considered low-power applications are currently commonplace and widely available. As a result, there is increasing impetus, as shown by technological advancements, ecosystem expansion, and the requirement for benchmarking and assessment approaches. We emphasize both challenges and opportunities and we provide an overview of the current state of the art in this field.

By providing ML inference to extremely low power, often a milliwatt or less, to devices with constrained storage, TinyML [15] aims to break the conventional power barrier that restricts globally distributed machine intelligence. TinyML increases privacy protection and responsiveness while lowering energy costs associated with wireless transmission, which at this scale are substantially more than those of computation, by doing inference near to the sensor and on the device [16]. A new class of intelligent, always-on, battery-powered apps that have the potential to revolutionize real-time data collection and processing is also made possible by efficiency of the TinyML. This rapidly expanding sector, which is the result of various innovations, is only expected to develop more quickly in the upcoming years [17].

IoT solutions that can lessen the burden of frequent data access and transfer to the cloud are necessary as the volume of data produced by IoT devices increases. Integration of TinyML into IoT devices is one way to address these drawbacks, according to source [18] TinyML is a framework that does not require constant connection to overloaded cloud services and runs on MCUs inside IoT devices. IoT ML services can therefore be provided to IoT devices or processed inside the IoT device in the current context [19]. The first approach is conventional, relying on the edge and cloud for the IoT-based system to provide ML services [19]. The second approach, in contrast, integrates TinyML into a contemporary framework to give IoT devices intelligence, which suggests that the MCU that interacts with the sensors may run the ML algorithm locally to forecast sensor data. Although the integration of ML into MCUs is mostly unexplored, first findings from the incorporation of the TinyML framework into IoT devices suggest that the area offers significant promise for ML execution at the very edge of the IoT. The next phase of hyper-digitization is thought to be TinyML.

To fit into the constraining limitations of MCU-class devices, TinyML models in particular must be compact enough (for example, constrained embedded computing capability of the order of MHz of processor clock frequency and a few hundred kB of memory), which restricts the input size and number of layers [11] or calls for the use of lightweight methods that do not rely on neural networks [20]. A few examples of TinyML tools are effective inference tools [21,22], frameworks (TensorFlow), memory-conscious neural architecture searches [10], and aggressive quantum methods [23]. The next generation of general-purpose MCUs will have more documentation, hardware intended specifically for low-power inference will be built, and new architectures will be created just for task-specific inference engines, according to research on TinyML hardware [13,24].

The intricacy and dynamics of the field conceal progress measurement and make dynamism planning decisions harder. To enable continuous innovation, a fair and trustworthy comparison method is essential. Increased hardware capabilities are typically the result of innovation, hence a reliable TinyML hardware evaluation is necessary. For many common

ML use cases, TinyML may seem futuristic. As ultra-low-powered inference hardware advances, the feasibility barrier rises [25]. Although object counting and image classification in a large label space are suitable for low-power continuous-operation applications, TinyML technology is still too computationally and memory intensive to handle these tasks.

The global market for edge computing is estimated to reach $1.12 trillion by 2023, according to [26]. IoT layer improvements must be aggressively pushed in order to lessen this burden, with a focus on ML in IoT devices. With companies like Ericsson already offering TinyML-as-a-Service solutions, the expanding need for smart devices has also been magnificently matched by the global research community [19]. With less reliance on cloud services, TinyML is anticipated to play a significant role in the provision of intelligent IoT solutions in the near future. It will additionally change the way IoT services are used in places with limited internet connection.

Additionally, TinyML will be crucial to next technologies. For instance, many of the crucial parts of augmented reality (AR) glasses are constantly active and powered by batteries. Such devices are unable to wait for calculations to upload to a cloud, edge server, or even another mobile device because of real-time constraints. As a result of frequent restrictions, developments in TinyML [27] potentially assist AR applications tremendously.

As businesses look for AI solutions in areas like environmental monitoring, people tracking, image or video control and voice activation, consumers have encountered difficulties due to the drawbacks of embedded modules and battery-powered sensors that perform with modest computing resources provided by general-purpose microcontrollers. Huge amounts of data usually need the use of sensors and edge devices. Additionally, due to their low power consumption, these devices struggle to maintain high data throughput and computational performance which causes latency problems. "Given that AI is being used to make quick choices in areas like quality control, alarm management and surveillance, any lag in the system might result in machine downtime or slowness, which would cause significant damage or loss of production. By moving AI to the edge, possible risks and vulnerabilities including erratic connectivity and sluggish answers are reduced", a principal analyst at ABI Research, Lian Jye Su, stated as an example [28].

The most essential, of however, is not just hardware development, which speeds up TinyML democratization [29]. TensorFlow Lite, the development of open source microcontroller software by Google, and for-profit solutions from companies such as SensiML provide libraries and developer-friendly software tools, enabling more engineers to build AI models that can manage really edge applications. It is no longer sufficient to create capable and distinctive hardware. TinyML hardware producers must focus on establishing new AI development environments or joining those that already exist, embracing open source, and highlighting to customers their distinct selling points and niche markets. In the absence of such conditions, chip suppliers may struggle to expand their goods in what is expected to be a highly competitive business [28].

2.2. Challenges

When it comes to constructing a performance benchmark that can be used to properly quantify and analyze performance differences between various systems, TinyML systems provide a special set of challenges. The several critical issues pertaining to the TinyML ecosystem are covered in this section [30,31]:

2.2.1. Low Power

TinyML systems are identified by their low energy usage. As a result, an effective benchmarking should theoretically define the energy efficiency of each device. However, accurately quantifying energy use presents several obstacles. For starters, TinyML devices can consume vastly varying amounts of energy, making it difficult to maintain accuracy over a wide range of devices.

When data pathways and preprocessing processes differ significantly between devices, it may also be challenging to define what is covered by power measurement. The measure-

ments may be impacted by additional elements, such as chip peripherals and underlying firmware. TinyML systems do not have redundant cores as standard high-power ML systems do, making it easier to load the System-Under-Test (SUT).

2.2.2. Limited Memory

As a result of their modest size, TinyML systems frequently encounter memory issues. TinyML systems usually have resource limitations two orders of magnitude less than normal ML systems, which frequently have restrictions of a few GB, such as smartphones. One of the crucial factors that influences how a certain TinyML test is created is memory. Inference models for traditional ML benchmarks require significantly more memory (in gigabyte range) than TinyML devices can provide. This makes designing a benchmark suite more challenging since any extra might significantly increase power usage or make the benchmark too large to run. Each benchmark has to test a variety of hardware, hence the benchmark suite should contain different quantization and accuracy levels. To that purpose, a variety of standards should be used to guarantee that the diversity of the sector is effectively supported. As a result, a variety of standards should be used to guarantee that the diversity of the sector is effectively supported. Finally, generating the need for creativity in the optimization strategies applicable to each algorithm is a difficulty. Due to the nature of DL compression models, it is crucial that the performance parameters match the initial model [32].

2.2.3. Processor Power

When compared to cloud-based systems, MCUs like the ARM Cortex M series still perform relatively poorly, even when coupled with powerful CPUs. As a result, while transmitting data analytics from the cloud to the device, the quality of service can degrade. Parallel to this, a rising variety of software frameworks are trying to solve the issue of ML algorithm compression, leading to frameworks that are richer and more varied. A formal structure is not yet accessible, though.

2.2.4. The Machine Learning Is Extensive and Requires Resources

The uniform and seamless deployment of the ML cloud in the embedded system is compromised by the absence of specialized ML hardware. Additionally, it is highly challenging to make the program portable or compressible since an ML algorithm and a software runtime are created using a complex high-level language. The fact that the time-consuming ML activities are still carried out on the cloud is therefore evident. Therefore, it is not feasible to totally replace web services with integrated ML.

2.2.5. Heterogeneous Hardware

Despite being in their early stages, TinyML systems are already diverse in terms of their abilities, power, and performance. Devices span from general-purpose microcontrollers (MCUs) to unique designs, such as event-based brain processors (brain chips) or memory computing [33]. Because the system under test (SUT) may not have generally conventional characteristics such as a system clock or debugging interface, this heterogeneity causes a number of issues. Furthermore, standardizing performance findings across varied implementations is a significant difficulty. Modern benchmarks are not built to solve these issues simply. They require careful redesign in order to be flexible enough to accommodate the kind of hardware heterogeneity seen in the TinyML ecosystem [34].

2.2.6. An Absence of Suitable Datasets

Due to a lack of low-power adaption, current datasets might not be appropriate for the TinyML paradigm. Such datasets ought to be precise enough in time and space to correspond to the properties of data generated by various sensors. Furthermore, this low-power edge device facility should be connected to the variety and noise level of such a

dataset. As a result, we emphasize the significance of a consistent dataset that can be used to train the TinyML system.

2.2.7. Network and Data Administration

TinyML development is insufficient if the data and network are not effectively controlled. It has been discovered that the existing edge network does not recognize diverse data types, resulting in less intelligent performance. Linked to a misunderstanding of the data lifecycle for sensor-equipped devices. The inconsistency of sensor data structure alleviates the challenge of edge data management. To overcome this issue, several options are possible. For example, data and sensor fusion methods can be extended to control the network at the edge. In this perspective, one can examine at the relevance of network issue detection and diagnosis. Augmentative learning algorithms must be combined with knowledge extraction approaches to improve knowledge sharing capabilities. Lightweight machine learning technologies might be investigated to promote autonomy and self-adaptivity in edge network management.

2.2.8. New Machine Learning Models Are Required

For the TinyML market, new machine learning models are now necessary. These models should be able to react extremely quickly. We may use reinforcement learning, federated learning, online learning, and transfer learning while learning utilizing the knowledge distillation dimension. For real-time solutions, machine learning models must include components for communication and control. Quantization and model trimming should be required steps in the model development process. Additionally, it should be proven that using edge devices results in total cost savings.

2.2.9. Benchmarking

TinyML has shown to be an important tool in the creation of next generation systems and is utilized in a wide range of applications over many industry sectors. This has enhanced the needs for TinyML system functional design and testing as well as led to its widespread adoption in a number of applications. System efficiency, real-time performance and power efficiency are crucial for getting the greatest performance out of TinyML applications and systems. Considering the significant breakthroughs being made in this sector by business and research, benchmarking of these ML applications and systems is crucial. These standards enable the community to appropriately evaluate and contrast solutions [35,36].

Our major focus on metrics and methodologies for assessing TinyML, rather than test chips and systems, is a significant problem we highlight. This issue involves multiple-level TinyML deployment capabilities in systems, engineering problems, and deployment design strategies. Research on robust TinyML architectures for various sizes, challenges associated with the cohabitation of non-ML and ML hardware, and TinyML-specific research for accelerator design are all of interest (embedded and microcontroller systems and systems, IoT and bare-metal sensors).

In conclusion, benchmarking TinyML systems exposes a number of issues and potential. We are interested in metrics and methodologies for assessing TinyML systems, for instance. We are also interested in metrics and assessing the efficacy of constructing such systems at various scales. We should be capable of comparing and evaluate new design approaches, methods, and methodologies like accelerators with reconfigurable and neuromorphic accelerators, approximate computational units, hardware and software co-configuration, model compression, in systems with both inference and non-ML hardware, and others. Furthermore, we must be able to transparently include these evaluation approaches and metrics into our design frameworks, allowing the designer to analyze a TinyML system holistically [37].

2.3. Academic Map

We locate our information using Google Scholar (https://scholar.google.com/). In addition, we carefully search for papers written by prominent TinyML experts. Below is a description of the search strategy (each term for each instance independently):

1. For the first search cycle, the term "TinyML" is utilized.
2. The term "Tiny ML" is applied to the second search cycle.
3. The term "Tiny-ML" is utilized for the third search cycle.
4. The term "Tiny Machine Learning" is used in the fourth cycle of searches.
5. The fifth search cycle use the term "Tiny Deep Learning".
6. For the sixth search cycle, the term "Tiny-DL" is utilized.
7. The seventh search cycle used the term "TinyDL".
8. For the eighth search cycle, the term "Tiny DL" is utilized.

For a comprehensive search, the 6 main databases are also used as supplementary sources in the research:

1. SAGE (https://us.sagepub.com/en-us/nam/home)
2. MDPI (https://www.mdpi.com/)
3. Wiley-Blackwell (https://onlinelibrary.wiley.com/)
4. Emerald (https://www.emeraldinsight.com/)
5. Elsevier (https://www.sciencedirect.com/)
6. Taylor and Francis (https://www.tandfonline.com/)
7. Springer (https://www.springer.com/)

But in order to give more comprehensive information, the chosen studies also contain different types of publications, such as reports, online journals and working papers, in addition to academic literature. Following that, the publication is chosen in two stages: First, by reviewing the title, abstract, and key words; and second, by reading the entire content. The year of publication is not a factor in selection, despite the fact that articles must be written in English due to a restriction on translation. Every source is available in paper copy or in downloaded form. The preliminary literature search produced 135 results (obviously the total number is different, it is derived based on the strategy followed as well as the time conducted). We arrive at 81 unique items for further study after analyzing the titles, abstracts, and keywords and deleting duplicates. 34 papers are discarded after comprehensive re-evaluation while reading the complete text, for a total of 47 articles (11 reviews/evaluations and 36 core research) utilized to collect the data. 26 conference proceedings, 15 journal publications (2 review articles and 13 research articles), 2 book chapters, 1 book, 1 newspaper article, 1 report and 1 working paper compose the review. The references are provided in detail in Table 1.

Table 1. Publications and references of various types.

Types of Publication	References
Conference Proceedings	[38–63]
Research Articles	[18,63–74]
Review Articles	[35,75]
Book Chapter	[76]
Books	[77,78]
Report	[79]
Working Papers	[80]
Newspaper Article	[19]

The model of the above process and some of its reports and results are based on the paper by Hui Han and Julien Siebert [81], where they provide a thorough survey of TinyML

and is a valuable source for scientific research on the subject. Future intention is to enrich it with several more papers as the scientific community will be more involved and TinyML will have even more growth in the upcoming years. Figure 3 showing schematically the percentages of the different types of references mentioned above.

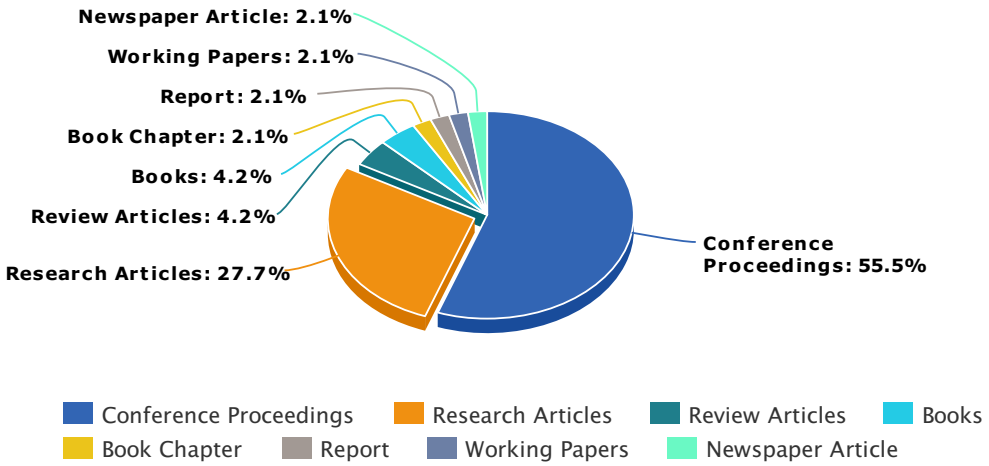

Figure 3. The different types of references.

2.4. Tools of TinyML

TinyML relies on a variety of software platforms, hardware requirements and libraries to make predictions. A summary of existing software and hardware toolkits being examined for prospective TinyML development is provided [30].

2.4.1. Hardware

Various platforms were selected which are aware of TinyML, such as: Nicla Sense ME: https://docs.arduino.cc/hardware/nicla-sense-me/, STM32F Discovery, ST IoT Discovery, Agora Product Development Kit, GAP9, Sony's Spresense TinyML Board, MKR Video 4000, FRDM-K64F, Pico4ML BLE, Nordic Semi nRF52840 DK, Himax EW-I Plus, OpenMV Cam H7 Plus, Nordic Semi Thingy:91, Thunderboard Sense 2, GAP8, Arduinoa Portenta H7, Raspberry Pi 4B, Apollo3, CC1352P Launchpad, ESP-EYE, XCore.ai, Seeed Wio Terminal, Arduino Nano 33 BLE Sense, Nvidia Jetson Nano, ECM3532 AI Sensor Neuro sensor processor (NSP) and AI-deck 1.1.

The Tables 2 and 3 are provided to compare the aforementioned hardware platforms in terms of SRAM capacity, CPU clock frequency, flash storage, processor, product programmer, connection, sensors or connectors, and power or voltage consumption. The majority of hardware boards operate at CPU frequencies under 100 MHz and typically include less than 1 MB of SRAM and 1 MB of flash. The most widely utilized connection technologies are Wi-Fi and Bluetooth (BLE). The majority of the boards include sensors including a light sensor, air pressure sensor, microphone, temperature sensor, humidity sensor, gyroscope, gesture sensor, accelerometer, air quality sensor, camera and hall-effect sensor. These boards require mW-level amounts of power. The majority of the gadgets can be powered by coin batteries and Li-Po in addition to a standard DC power source. Among all options, the most popular CPU is ARM Cortex-M4. A hardware coalescing engine (HCE) is incorporated into just few devices (for example, GAP8, GAP9).

Table 2. Comparison of TinyML Supporting Hardware Platforms (CPU Clock < 100 Mhz).

Board/Platform	Micro-Processor	CPU Clock Speed	Flash Memory	SRAM Size	Voltage and Power	Connectivity Availability	Connectors and Sensors	Company
Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT Kit	EFR32™ Mighty Gecko Wireless SoC	38.4 MHz	1 KB	256 KB	3.3–5 V, Coin cell, ULP	SPI, 2.4 GHz, USB	Pressure, air quality, microphone, temperature, humidity, ambient light, hall-effect, UV	Silicon Labs
Syntiant Tiny ML Board	Syntiant® NDP101 NDP, 32-bit ARM Cortex-M0	48 MHz	256 KB	32 KB	3.7–5 V, LiPo battery	I2C, UART	Microphone, motion	Sytiant
TI CC1352P LaunchPad	CC1352R Wireless MCU LaunchPad™	48 MHz	352 KB	8 KB	60 µA/MHz, 1.8–3.8 V	868/915/433 MHz, UART, ZigBee, SSI, I2C, Thread, I2S, BLE, 802.15.4, Sub-1 Ghz	Temperature	TI
MKR Video 4000	Intel® Cyclone® 10CL016 FPGA, 32-bit ARM Cortex M0	48–200 MHz	2 MB, 256 KB	32 KB, 8 MB SDRAM	3.7 V Li-Po, 1024 mAh	USB, MIPI, u-blox NINA-W102, SPI, UART, I2C	–	Arduino
Apollo3	32-bit Arm® Cortex®-M4F	48 MHz, 96 MHz with TurboSPOT™	1 MB	384 KB	6 µA/MHz Battery option,	FTDI SPI, USB, BLE5	MEMS microphone, accelerometer, HM01B0 camera	SparkFun
STM32F Discovery	32-bit Arm® Cortex®-M4 FPU Core	48 MHz	1 MB	192 KB	3–5 V	USB, LQFP100 I/O	Microphone, accelerometer	STMicroelectronics
ST IoTDiscovery	Arm® Cortex®-M4	48 MHz	1 MB,64 Mbit Quad-SPI	128 KB	Battery option	868/915 MHz, BLE 4.1, NFC, USB, 8.211b/g/n	Accelerometer, microphone, gesture detection, gyroscope, temperature, barometer, humidity	STMicroelectronics
Nordic Semi nRF52840 DK	Arm® Cortex®-M4	64 MHz	192 KB	24 KB	1.7–5 V Li-Po	Zigbee, BLE5, NFC, Thread, UART, 802.15.4, ANT, 2.4 GHz, Bluetooth mesh	–	Nordic
Arduino Nicla Sense ME	Arm® Cortex®-M4	64 MHz	512 KB	64 KB	3.7 V Li-Po	USB, BLE4.2, I2C, SPI	Geomagnetic, accelerometer, gyroscope, humidity, pressure, geomagnetic, gas, temperature	Arduino
Nordic Semiconductor Thingy:91™ Multisensor Prototyping Kit	Arm® Cortex®-M33, nRF9160 SiP	64 MHz	1 MB	256 KB	1440 mAh Li-Po	I2S, SPI, LTE-M, NB-IoT, UART,	Humidity, pressure, color, air quality, temperature, light	Nordic
Arduino Nano 33 BLE Sense	nRF52840	64 MHz	1 MB	256 KB	3.3 V, 15 mA/pin	USB, UART, SPI, I2C, BLE, SPI	Barometer, IMU, gesture, temperature, light, proximity, humidity, microphone	Arduino

Table 3. Comparison of TinyML Supporting Hardware Platforms (CPU Clock $\geq$ 100 MHz).

Board/Platform	Micro-Processor	CPU Clock Speed	Flash Memory	SRAM Size	Voltage and Power	Connectivity Availability	Connectors and Sensors	Company
ECM3532 AI Vision Board	Arm® Cortex®-M3, NXP CoolFlux 16-bit DSP	100 MHz	512 KB	256 KB	5 µA/MHz, Battery option	USB, RF, BLE 4.2	Temperature, pressure, microphone, gyroscope, accelerometer	Eta Compute
Freedom-K64F	Arm® Cortex®-M4	120 Mhz	1 MB	256 KB	1.7–3.6 V, Coin cell	CAN, I2S, SPI, I2C, UART, Ethernet	Magnetometer, accelerometer	Mbed
Arducam Pico4ML-BLE	Raspberry Pi RP2040 DSP dual core	133 MHz	4 MB	264 KB	1.7–3.6 V, battery	I2C, USB, BLE	IMU, camera QVGA 60 FPS, microphone	ArduoCam
Sony's Spresense	Arm® Cortex®-M4F 6 Core	156 MHz	8 MB	1.5 MB	3.3–5 V	GNSS antenna, UART, I2C, SPI, I2S	Camera, microphone	Sony
AI-deck 1.1	GAP8, ESP32	168 MHz	1 MB	192 KB	3–5 V	URT, SPI, WiFi	Monochrome camera	Bitcraze
ESP-EYE	32-bit ESP32	240 MHz	4 MB	8 MB PSRAM	3.3 V	UART, USB, BLE, SPI, I2C, WiFi	2MP camera	Espressif
GAP8	RISC-V, hardware convolution engine	250 MHz (FC), 175 MHz (C), 22.65GOPs	512 KB	80 KB, 8 MB SDRAM	1.8–3.3 V, 4.24 mW/GOP	I2S, SPI, UART, I2C, CPI, Hyperbus, Serial	Extension camera	Green Wave Technologies
Himax EW-I Plus	32-bit ARC EM9D DSP with FPU Core	400 MHz	2 MB	2 MB	1.2–3.3 V, Battery	USB, SPI2, UART, I2C	Accelerometer, VGA Camera 60 FPS, microphone	SparkFun
GAP9	RISC-V, hardware convolution engine	400 MHz, 150.8GOPs	1.5 MB	128 KB, 2 MB External	1.8–3.3 V, 0.33 mW/GOP	CPI, SPI, I2C, UART, I2S, Hyperbus, Serial	Extension camera	Green Wave Technologies
Arduino Portenta H7	Arm® Cortex®-M7, Arm® Cortex®-M4 GPU	480 MHz, 240 MHz	16 MB	8 MB SDRAM	3.7–5 V, Li-Po cell, 700 mAh	MIPI DSI, BLE, 10/100 Ethernet Phy, USB, MPI D-PHY, WiFi	Camera extension, temperature	Arduino

Table 3. *Cont.*

Board/Platform	Micro-Processor	CPU Clock Speed	Flash Memory	SRAM Size	Voltage and Power	Connectivity Availability	Connectors and Sensors	Company
OpenMV Cam H7 Plus	Arm® Cortex®-M7	480 MHz	2 MB (Internal)	1 MB, 32 MB SDRAM	3.7 V Li-Ion	UART, USB, I2C, CAN	5MP Camera at 50 FPS	OpenMV
XCore.ai	Convolution and dense neural network FPU 16 core	3200MIPS, 1 M 512 FFTs/s	–	1 MB	1.8–3.3 V, 500 mW	USB, MIPI, I2C, UART, SPI, I2S	–	XMOS
Raspberry Pi 4 Model B	64-bit Arm® Cortex®-A72 quad core, Broadcom BCM2711	1.5 GHz	–	256 KB	3.8–4 W, 3.3–5 V	Ethernet, USB, HDMI, WiFo, BLE, DSI, CSI	Temperature	Raspberry Pi

2.4.2. Software and Libraries

A number of platforms, frameworks and libraries were selected that are up-to-date with TinyML technology, such as:

- *TensorFlow Lite (TFL)*: It is a deep learning framework that is open source and supports edge-aware learning inference. This framework may approach edge-aware machine learning at the device by using five important restrictions (e.g., size, latency, connectivity, power consumption and privacy). It is compatible with iOS, embedded Linux, Android, and a range of microcontrollers [82]. Swift, Java, C++, Objective-C, and Python are just a few of the programming languages that are supported for machine learning on edge devices. TFL enables hardware-accelerated model optimization. A wide range of AI applications, including photo and text categorization, question response, object identification, and pose estimation, may be easily handled. Because all operators are coupled to 32-bit ARM builds, the binary size is 1 MB. When particular image classification techniques are used, it can generate a binary file as little as 300 KB. The TFL working process is completed by quantizing the 32-bit floating point values to 8-bit integers, which completes the TF model transformation into a compressed flat buffer (.tflite) and loading into an embedded edge device. A TFL plugin called TensorFlow Lite Micro (TSFM) was created to enable machine learning on ARM Cortex processors with KB memory. TFLM runs on a 32-bit platform and was created in C++ 11. However, it does not offer on-device training.
- *NanoEdge AI Studio*: Previously known as cartesiam.ai, the software today tests library performance using an emulator before final deployment to the edge and allows for the selection of the best library [83]. It contains various useful features, such as (i) frequency filtering, (ii) restricting the maximum necessary flash memory when creating a project, (iii) flash memory optimization, (iv) real-time search, (v) graphical representation of serial data, and (vi) library selection after comparison. It may be used to discover and classify abnormalities in data sets. It is compatible with the Arduino Nano 33 IoT board as well as the STM32 Nucleo-32 board.
- *PyTorch Mobile*: Belongs to the PyTorch ecosystem, which aims to make it possible for all stages of machine learning model generation, starting with training (for example, Android, iOS), can be done on smartphones and tablets [84]. Machine learning may be pre-processed in mobile apps using a variety of APIs. Both TorchScript IR detection and scripting are supported. Additionally, the 8-bit quantized XNNPACK kernel is supported for ARM CPUs. Additionally supported are neural processing units, digital signal processors and GPUs. The mobile interpreter enables mobile development optimization. Now supported are question answering, speech recognition, object identification, video processing, and image segmentation.
- *uTensor*: It is an open-source embedded learning environment that facilitates rapid development and prototyping on IoT edge devices [85]. It includes a future data collection architecture, a graph processing tool and an inference engine. For training, a Keras-made neural network model is required. After that, the learnt model is translated into C++. The model is modified for usage on ST, K64, and Mbed boards with the aid of uTensor. With just 2 KB of storage needed, the uTensor is a tiny device. The use of a Python SDK is required for full configuration of uTensor; it depends on

the Jupyter, Python, ST-link toolkits (for ST boards)—uTensor-CLI and Mbed-CLI. A model is built initially, and then the quantization outcome is produced. Writing code for the proper edge devices is the next stage.

- *STM32Cube.AI*: This is optimization software and code creation for STM32 ARM Cortex Mb-based boards [86], trying to make machine learning and AI-related jobs simpler. The STM32Cube may be used directly to implement neural networks on the STM32 board. AI will be used to turn neural networks into efficient code for better-suited MCUs. It is capable of employing any trained model produced using conventional tools like MATLAB, ONNX, TFL, and PyTorch. The STM32CubeMX framework that supports STM32Cube originally gave birth to this utility. AI in parameter estimation middleware and code development for the STM32 edge device.
- *Edge Impulse*: For edge computing systems, it is a cloud service that develops TinyML machine learning models. For edge platforms, it can support AutoML [87]. The building of learning models is also supported across a number of platforms, including smartphones. The learning is done in the cloud, and using a data-forwarding-capable connection, the learned model may be exported to an edge device. Using the integrated Python, Node.js, C++, and Go SDKs, it may be executed locally on a workstation. A WebAssembly library for Impulses is also available.
- *Embedded Learning Library (ELL)*: Microsoft released ELL to enable the TinyML ecosystem for embedded learning [88]. It works with the micro:bit platforms, Raspberry Pi and Arduino. Models built on such devices are internet-independent, therefore no connectivity to the cloud is necessary. Image and audio categorization are presently supported.
- *µTVM*: Tensor programming can be conducted on microcontroller devices thanks to the microTVM extension of virtual tensor machines (TVM) as it is currently. The AutoTVM platform, which supports the optimization of tensor programs, makes it possible to optimize these programs [89]. In fact, a USB-JTAG interface links a microcontroller to a computer or high-end device that is simultaneously running the TVM. The PC runs OpenOCD, which connects the microcontroller to the computer. By applying device identification to the TCP port, OpenOCD allows mTVM to operate the microcontroller. The user must submit specifics (such as a method for reading, writing, and executing to the memory of the device, a C cross-compiler toolchain for a microcontroller, a description of the architectural layout of the device, and a section of code to set the device up for operation) in order to receive support from µTVM. The MicroSession must connect to the device using the provided way in order for the µTVM to function (e.g., OpenOCD). The previously mentioned cross-compiler is then used to cross-compile the µTVM runtime. The binary of the produced code is then transferred to the device. The relationship between the µTVM and TinyML may be discussed from several angles, including tensor loading, unit loading function calling, and slow execution. In Figure 4, the system model based on TinyML [90] is shown for building the best models on microcontrollers, sometimes referred to as edge devices.

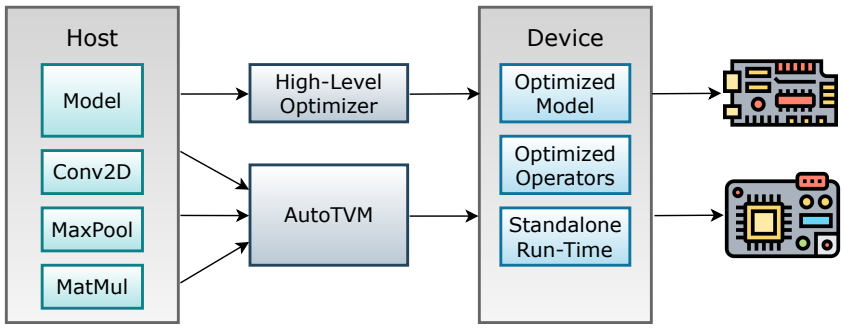

Figure 4. µTVM optimization and application on a microcontroller system model.

The subsequent paragraphs of this section will provide a quick overview of a number of TinyML frameworks and libraries. Relevance Vector Machine (RVM) and Support Vector Machine (SVM) algorithms may be translated to C code and operate on a number of MCUs, including the ESP8266, ESP32, Arduino, and others that support C. The effort known as MicroML makes this transition possible. By working with the well-known Scikit-learn toolbox, the models developed by this library are changed so they may be run on 8-bit microcontrollers with 2 kb of RAM. Additionally, trained estimators may be exported to JavaScript, Java, C, PHP, Ruby, and GO using the Scikit-learn compatible program sklearn-porter. This is a relatively inclusive framework that is also compatible with a broad variety of machine learning approaches, given the variety of supported languages. The sklearnporter can translate Scikit-learn models for the C programming language, such as Random Forest (RF), SVM, the AdaBoost classifier, and decision trees. However, due to the flexibility of the library, it cannot generate code for microcontrollers with the least amount of RAM requirements. Similar software, m2cgen, may transform taught Scikit-learn models into native code, including Visual Basic, JavaScript, Python, C, Go, Java, and other languages. The sklearn porter is unable to handle the vast majority of compatible algorithms and target programming languages in that situation. The Weka-porter application, which has less capabilities, utilizes WEKA decision tree classifiers made with Java, JavaScript and C code. Scikit-Learn or WEKA models may be converted into C++ source code files that can be produced and executed on a constrained number of systems using the EmbML package. This library has been tried and tested on Teensy and Arduino boards and supports several machine learning (ML) methods, including decision trees, SVMs, and RNNs [18].

Python libraries like Keras and Scikit-learn are converted into portable C99 code via the emlearn framework. This library has been tested on a number of platforms, including the Linux, AVR Atmega and ESP8266, and is compatible with models created using several techniques, including as Naive Bayes, decision trees, RF, and linear models. TinyMLgen is a library that generates ready-to-run C code that can be included into various MCU families, with the goal of enhancing TensorFlow model interoperability. The first is centered on Mbed boards, but the later outputs the trained model in a simple C table, allowing it to be used on a larger range of microcontrollers.

Last but not least, two research ideas focus on enhancing how different NN types are executed on ARM Cortex-M processors. Quantized NN is implemented on MCUs using an open-source mixed-precision library called CMix-NN. This utility provides convolution kernels with 2, 4, or 8 bit precision for any of the operators. In their recent investigation, the scientists utilized the convolutional NN models created by MobileNets. Additionally, the open-source framework FANN-on-MCU, which aims to run lightweight neural networks on microcontrollers, is based on the Fast Artificial Neural Network (FANN) library [91]. That toolkit in particular can work with Multi Layer Perceptrons (MLPs) trained using FANN and generate code that can operate on parallel ultra-low power (PULP) devices in addition to the ARM Cortex-M CPU discussed above.

Tables 4 and 5 summarizes the key characteristics of the frameworks and libraries covered in this section while also giving a market perspective.

Table 4. TinyML Frameworks & Libraries.

Framework	Algorithms	Compatible Platforms	Publicly Available	Main Developer
emlearn	Random forest Decision tree Naive Gaussian Bayes Neural networks	AVR Atmega ESP8266 Linux	Yes	Specific developer
EmbML	SVM Decision tree Neural networks	Arduino Teensy	No	Research group
weka-porter	Decision tree	Nonconstrained platforms & multiple constrained	Yes	Specific developer
TinyMLgen	Neural networks	ARM Cortex-M ESP32	Yes	Specific developer
uTensor	Neural networks	mBed boards	Yes	Specific developer
FANN-on-MCU	Neural networks	ARM Cortex-M PULP	Yes	Research group
CMix-NN	Neural networks	ARM Cortex-M	Yes	Research group

Table 5. TinyML Frameworks & Libraries (Continued).

Framework	Algorithms	Compatible Platforms	Publicly Available	Main Developer
MicroMLGen	SVM RVM	Arduino ESP32 ESP8266	Yes	Particular developer
MicroMLGen	SVM RVM	Arduino ESP32 ESP8266	Yes	Particular developer
m2cgen	LGBM Classifier Logistic regression Linear regression SVM Neural networks Decision tree Random Forest	Multiple constrained & nonconstrained platforms	Yes	Particular developer
AIfES	Neural networks	ARM Cortex-M4 Windows (DLL) STM32 F4 Series Arduino ATMega32U4 Raspberry Pi	No	Fraunhofer IMS
CMSIS-NN	Neural networks	ARM Cortex-M	Yes	ARM
ELL	Neural networks	ARM Cortex-M ARM Cortex-A Arduino micro:bit	Yes	Microsoft
TensorFlow Lite	Neural networks	ARM Cortex-M	Yes	Google
ARM-NN	Neural networks	ARM Ethos Processor ARM Mali Graphics Processors ARM Cortex-A	Yes	ARM
STM 32Cube. AI	Neural networks	STM32	Yes	STMicroelectronics

Table 5. *Cont.*

Framework	Algorithms	Compatible Platforms	Publicly Available	Main Developer
sklearnporter	Neural networks SVM Random Forest Ada Boost Classifier k-NN Decision tree Naive Bayes	Multiple constrained & nonconstrained platforms	Yes	Particular developer
NanoEdge AI Studio	Unsupervised learning	ARM Cortex-M	No	Cartesian

The Figure 5 shows the percentage used by each algorithm of the total number of frameworks analyzed previously.

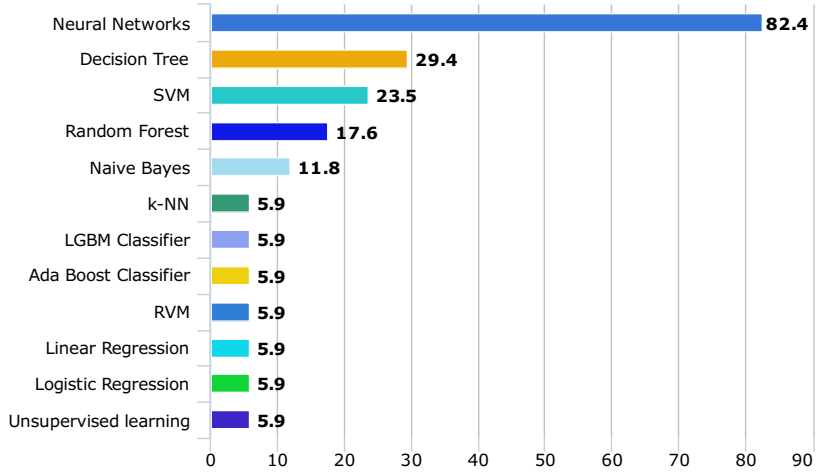

Figure 5. Percentage of use of different algorithms from the total of the frameworks previously analyzed.

2.5. TinyML Benefits

TinyML performs calculations near to the sensors, which could also improve or even introduce new data processing approaches that were previously inaccessible in low-resource settings. Intrinsic qualities of the TinyML, such as flexibility, efficiency, and simplicity, demand consideration due to their potential to change the entire IoT ecosystem. The following are the main performance metrics that demonstrate effectiveness of the TinyML as a key tool [31]:

i. *Transition from basic to smart IoT devices*
The capacity of the sensor system to generate massive volumes of raw data has also hindered the ability of cloud computing to process these data. Due to the lack of transmission to the cloud, the abundance of data is wasted at the edge. TinyML enables data analysis in a resource-constrained setting, where every IoT device becomes smart by embedding ML algorithms. A smart car, for instance, creates 1 GB of data per second [92], whereas the Boeing 787 generates 5 GB per second [93]. Therefore, it would be better to do a preliminary analysis of raw data using ML algorithms on each IoT device in order to gather necessary information and eliminate useless data rather than uploading all raw data to the cloud.

ii. *Network Bandwidth*
In order to input data, evaluate information, and execute ML algorithms, the "traditional" IoT uses gateways, a network of sensors to gateway networks, cloud services and gateways to the internet network [94]. In contrast, novel techniques of the TinyML have the ability to redefine these criteria in resource-constrained

settings by lessening the pervasiveness of cloud services and making other IoT layer services optional.

TinyML offers greater independence than standard IoT services thanks to its low transmission and maybe constrained bandwidth capabilities. Furthermore, in a setting with a high density of IoT devices, the bandwidth needed for providing raw data is rather significant. Therefore, analyzing raw data at the edge and only transmitting the information that is essential would significantly minimize the needed bandwidth.

iii. *Security and privacy*

As enormous amounts of private data are transferred to the cloud, data security is one of the variables impacting IoT adoption [95]. When third-party suppliers are employed for IoT services, the end user has no idea who owns the data or where their personal information is maintained. Additionally, data transfer makes it simpler for shady individuals to eavesdrop. Therefore, by avoiding data leakage and keeping data confined within the device, privacy and security are strengthened. Since TinyML data is rarely (or never) delivered, it is less vulnerable to attacks. Because of this, TinyML by default has built-in data security and privacy protections.

iv. *Latency*

Once the sensor data is sent from IoT devices to cloud servers, the decision (prediction) generated in the cloud by the IoT devices comes to an end in an IoT ecosystem. This series of events clearly shows that detailed observation of the device is required because to the high latency of the strategy. TinyML is a good way to deal with this issue. Additionally, on safety-critical systems like healthcare (such as microsurgery) and driverless cars, waiting for the cloud to decide might have disastrous consequences. As there is less reliance on external connectivity in such cases, TinyML will act as an infrastructure to enable decreased (near-zero) latency for ML service delivery. Local real-time data processing on the devices enables quicker reaction and analysis in emergency situations. Furthermore, the strain on the cloud is minimized [96–98].

v. *Energy efficiency*

Another major and popular TinyML indicator in MCUs is this one. The majority of IoT devices operate on batteries and are constantly on in an IoT ecosystem. Coin batteries, like the CR2032, are frequently used to power IoT devices, and they should allow them to operate for several months or perhaps several years. To enable the MCU to review the data and power on when necessary, it is usually crucial that the device remain predominantly in a suspended state. Data transport may occasionally use more energy than local ML service provision, too. TinyML would be a very useful tool for resolving these problems [99].

vi. *Reliability*

The capacity to do data-driven calculations within the sensor network is the main feature that is much wanted for IoT. TinyML has been recognized as a resolution for executing work in situations where mobile connectivity/internet is extremely restricted, such as offshore and rural areas. IoT services become more dependable as a result.

vii. *Low cost*

First off, by limiting data flow, bandwidth requirements are lowered, which results in cost savings. To achieve this, we believe that TinyML solutions paired with cloud technology can enhance the aforementioned performance metrics by adding new data management channels, expanding the capability of current cloud services, or delegating tasks to MCUs. TinyML is therefore the upcoming big thing, and many are touting it as the technology that will fuel the digitization of industries or the Industry 4.0 revolution [100].

viii. *Data Filtration*
The intelligence of the IoT device enables the designer to examine the data and remove residuals. When deployed in use cases with high traffic volumes, the intelligent support system at the edge can significantly outperform traditional systems. A surveillance system that focuses on anomaly detection, for instance, consists of a number of cameras, and the vast majority of the data that the cameras collect is redundant. In these cases, it makes more sense to filter the extra photographs locally rather than uploading them all to the cloud.

2.6. Applications Areas of TinyML

In fact, the vast array of applications that the development of TinyML has made possible is one of the driving reasons behind the extraordinary growth of this movement. The applications generate the essential data sets and corresponding economic growth that feed the advancement of research, driving the demand for more effective TinyML systems and additional improvements in the overall design process. The previous two years have seen TinyML applications steadily grow from their initial focus on vision to encompass natural language processing, predictive modeling, pattern recognition/classification, and data analysis. An overview of the application areas is given together with a high-level analysis of the challenges and opportunities.

TinyML is presently being used efficiently in a variety of applications, including smart objects (from sensors to cities), industrial control and monitoring, healthcare, surveillance and security, administration and finance and several others covering our daily lives and communities [18,32,101–104]. Below is a small sample:

1. Intelligent Objects (IoT):
Always-on wake-up units (voice, activity and motion [105]) such as translation and communication applications, smart assistive devices, natural language processing, intelligent communication systems [106], virtual and augmented reality, personalized and user-adapted services, etc.
2. Monitoring and Control of the Industry:
TinyML has considerable potential influence on the industrial and manufacturing sectors, which are undergoing digitization as part of the Industry 4.0 revolution [100]. MCUs are unable to consistently carry out some operations because of the highly changing compute and memory requirements of the processing activities in these sectors. The incorporation of ML-based decision support systems (DSS) into MCUs, on the other hand, enables them to choose whether to take on a given computational task or offload it to higher processing layers, such the edge or cloud. Furthermore, intelligence along the production chain may be used to improve production processes, decision making, asset monitoring, production and assembly line quality assurance, real-time diagnostics of assembly machines, and so on [107].
3. TinyML in Healthcare:
Wearable devices, such as smartwatch, are loaded with a variety of biological sensors that monitor vital functions like as heart rate, blood oxygen concentration, exercise, and even obesity [108,109], and deliver accurate real-time visualization of the current health status of the user in a confidential, safe, and dependable manner. Smart camera sensors capable of monitoring patients in their own environments and swiftly assisting with nurse notification, real-time diagnosis and aid, personalized and translational medicine, and so on [110,110,111]. Additionally, smart microprocessors that utilize TinyML, can effectively predict the likelihood of an accident [112].
4. TinyML in Security and Surveillance:
Camera sensors with hardware capable of conducting relatively rapid and accurate visual processing are used in a variety of industries, including tele-health [113], security and surveillance, services, surveillance, navigation devices, and so on.
5. TinyML in Smart Agriculture [114]:

TinyML offers a lot of potential in Africa, where embedded systems and artificial intelligence are still underutilized [115,116]. Cassava farming is one of these chances, as it provides a key source of food for hundreds of millions of people each year. However, it is always under threat from many illnesses. To combat it, PlantVillage [117], an open-source project managed by Penn State University, has created Nuru, an artificial intelligence-based program that can function on mobile phones without internet connectivity—a valuable asset for distant African farmers. By evaluating sensory data in the field, the Nuru app has been successful in minimizing hazards to cassava farming. PlantVillage intends to employ TinyML more widely in the development of Nuru, sending microcontroller sensors to remote farms to offer better monitoring information for analysis. TinyML is also discovering new applications in the agricultural commodities chain, such as coffee beans [118]. For example, two Norwegian businesses, Roest and Soundsensing, have devised a method to automatically recognize the "first crack" of coffee beans during the roasting process. It is critical to identify the first crack since the time spent roasting after the first crack has a major impact on the quality and flavor of the processed beans. To accomplish this task, companies have added a microcontroller with TinyML in their bean roasting equipment, which has increased the efficiency, precision, and scalability of the coffee roasting process. Also, a farmer could with the right equipment as well as with appropriate forecasting models running for weather forecasting, know locally for that particular field its daily weather and its characteristics.

6. TinyML in Vehicular Services and Autonomous Vehicles:
   As more environmentally friendly and healthful vehicles, such as shared bicycles, electric mopeds, scooters, and so on, come into existence, current patterns in urban transportation are changing. However, due to the high energy requirements of their linked on-board units (OBUs), modern vehicle networks largely take into account conventional forms of road transportation including cars, buses, and trucks. Although various attempts to connect bicycles to C-ITS systems have been made [119], personal autos have not been taken into consideration due to their novelty and inherent limitations. The ability to integrate these devices into C-ITS and smart city ecosystems, on the other hand, is made possible by connecting them with MCU-based OBUs [120]. Thus, basic vehicle services like route planning, device status monitoring, and driving safety and so on [120] will be accessible to light cars.
   Autonomous driving has made giant strides since the advent of deep learning (DL). Small vehicles driving choices have historically been off-loaded to remote computers, requiring energy-intensive, time-consuming, and unreliable transfers of raw data. By processing data on-board and immediately controlling the motor controllers, off-system transfers may be avoided. However, because the system is battery-powered, only a tiny portion of the electricity can be sent toward the processing unit, which is the autonomous vehicle's "brain". TinyML approaches are therefore required to solve these issues and deal with on-device sensor data processing at the hardware, algorithmic, and software levels.
7. TinyML in Smart and Secure Societies:
   Water systems, a safe food supply chain [121], intelligent transportation systems, disaster relief [122], smart energy grids [123], and emergency response technologies [124], and so on [125]. Also, efficient defect detection in modern production lines, such as logistics, in numerous stages of the manufacturing process [126].
8. TinyML in Intelligent New Spaces:
   The collaborative intelligence that now exists in scenarios like smart cities and cognitive buildings, to name a few, will be strengthened by simple objects with ML capabilities. Current IoT-based surveillance and monitoring systems, such as those used for traffic, pollution [96], and crowd identification, will grow into autonomous and intelligent entities [127] capable of making quick and decentralized choices. Because of independence from the power grid and the simplicity of installation, items

may be installed in isolated and rural locations, creating smart spaces [128] with smooth mobility between them. Furthermore, lower end-device costs will stimulate their adoption in underserved areas, which may assist revitalize local economies and commercial activity.

The aforementioned areas of application are obviously just a small portion of the potential of the TinyML, but they have served as the cornerstone for the technology's development by creating an ecosystem that (a) produces the datasets required to inspire the TinyML market to improve and create more useful products, (b) offers the services required by society to enable public acceptance without the associated mistrust that typically surrounds artificial intelligence, and (c) inspires through emerging technology [18]. Figure 6 depicts the major sectors that comprise the TinyML ecosystem.

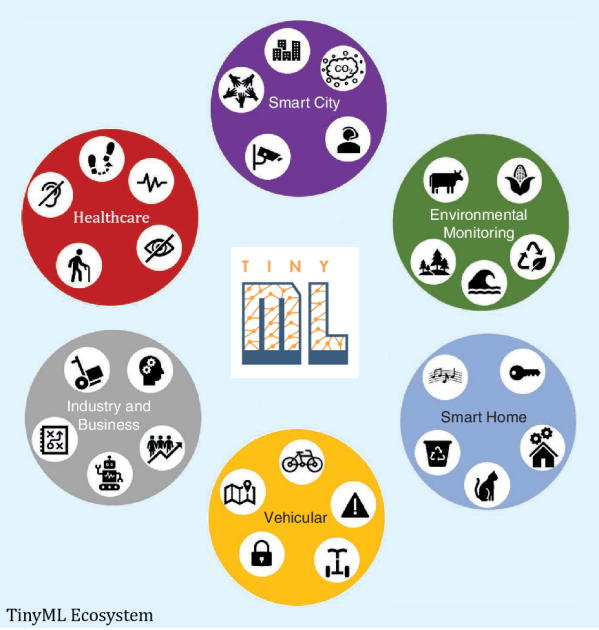

Figure 6. TinyML Ecosystem.

2.7. Use Cases of the TinyML

TinyML may be used to showcase a variety of use cases. Phenomenology, face detection, gesture recognition, keyword detection, sign language processing, precision agriculture, sea turtle protection, cough detection, posture estimate, posture evaluation, ecological monitoring, image recognition, traffic control, environmental forecast, autonomous vehicle monitoring, speech recognition, respiratory symptoms, and always-on voice wake-up are examples of use cases.

2.7.1. Image Recognition

In [91] involves visual awareness condensers to achieve deep image recognition. As a device-based framework for low-precision image identification, AttendNets is provided. It uses developed micro-macro-architectures that are supported by a design process that is driven by machines. $ImagNet_{50}$ is used as a benchmark to compare AttendNets against in order to demonstrate its relevance. According to the results, it decreases multiply-add processes by three times and increases accuracy by 7.2%. Additionally, it utilizes 16.7 times less weight memory than MobileNet-V1 and 4.17 times less parameters. The architecture provides the best possible balance between network efficiency and accuracy gap. By using automated micro-macro structures, it enhances selective spatial channel attention. On-device image recognition may become a critical facilitator of TinyML implementation for low cost and low energy sensitive IoT devices as a result of the usefulness of such integration.

2.7.2. Hand Gesture Recognition

In the TinyML realm, gesture recognition is a particularly promising subject. Various approaches for applying gesture prediction to resource-constrained devices are being investigated. For example, hyper-dimensional computing is being examined for the application of electromyogram pattern recognition [129]. The prospect of gesture prediction with limb position awareness utilizing sensor fusion technology is studied in this work. To replicate dual-stage categorization, the accelerometer and EMG data are combined. Multiple models keep position-specific properties in superposition. When evaluated on a dataset comprising 8 limbs and 12 motions, it achieved 93.34% accuracy during model validation. The findings suggest that this activity uses 8 times less RAM. The combination of EMG sensors and accelerometer sensors for hand signal collection is shown in Figure 7. In [108], an ANN model was used to detect and categorize a time series of a sensor readings for a collection of motions (Forward, Backward, Select and Stop). To conduct the experiment, they used an ATMega4809 CPU, a composite eye camera, and a hand gesture sensor. The microcontroller featured clock processing speed of up to 20 M, a 48 KB of flash memory, and 6 KB of RAM. They produced synthetic data after recording different actions made by a complete hand, arm, or fingers at different distances. Models included Feed Forward Neural Networks (FFNN) and RNN. FFNN outperformed RNN, according to the results of the model. With the ReLu activation function, FFNN had an accuracy of 84% whereas RNN had an accuracy of 83%. The 32 KB flash memory and 2 KB RAM of the ATMega328P microcontroller were verified to be adequate for achieving a frame rate of 40 Hz using both methods.

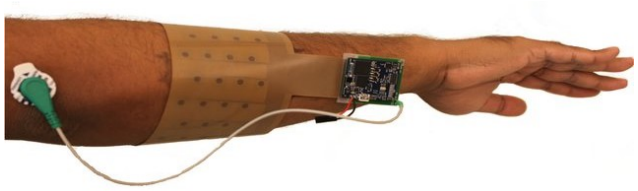

Figure 7. Wireless EMG + accelerometer acquisition device [30].

Capacitive sensing has been addressed for gesture prediction for TinyML configurations [105]. The embedded prototype created for this study contains four capacitive sensing electrodes and may be worn on the wrist. It has a classifier accuracy of 96.4% and uses just one hidden layer. An ARM Cortex-M4 MCU with 256 KB of RAM and 1 MB of flash is used in the design. Without quantization, the released embedded prototype in float32 mode is just 29.6 KB in size. The inference process requires 26.4 mW of electricity and takes only 12 ms. In a different study, it was discovered that the first finger was equipped with an NRF52 CPU and a 32 Hz accelerometer. From the TensorFlow Lite Micro library, it selects a multilevel long short-term memory (LSTM) model with 2.8 MB of on-disk capacity. With a measurement accuracy of 95.5% over 10 distinct movements, it is trustworthy.

2.7.3. Face Detection

A crucial application case for the smart IoT environment is face recognition. Long-range face detection without batteries, which is readily accessible, may surely enhance human face tracking in a number of applications. In [48], a time-of-flight sensor and a low-power camera module (such as the HM01B0) are paired with an ARM Cortex-M4F MCU (e.g., VL53L1X). The LoRa transceiver module is used to interact with a distant system (e.g., SX1262). The BQ25504 chip controls the system power. Initially, the flight time unit is engaged automatically (e.g., 1 Hz).The MCU is programmed to wake up if it notices any motion. Images captured by the camera are 320 × 320 pixels. There are VGA and QVGA options available with this configuration. At 30 frames per second, this uses just 1.1 mW of power. Two layers of convolution are utilized in this mode, along with two levels of convolution-MaxPool. Three dense layers help in facial detection in the end. The CelebA dataset is used in the article to train a face classifier on the MCU using the TensorFlow

framework. In the wireless camera detection planning mode, it achieves 97% accuracy. The practical IoT-based face detection system is depicted in Figure 8. For the COVID-19 to be introduced, face mask identification has been crucial. The creation of face mask recognition systems that are TinyML aware has been tried by a number of research teams. In [130], for example, a convolution neural network that performs facial recognition utilizing 496 KB of buffer memory is built using an ARM Cortex-M7 MCU running at 480 MHz. The model, which was trained using TensorFlow, requires 138 KB of RAM at 30 frames per second during post-quantization. In [131], a visual face device that detects coughs is displayed. When a cough is detected, the face visor may be switched on and off.

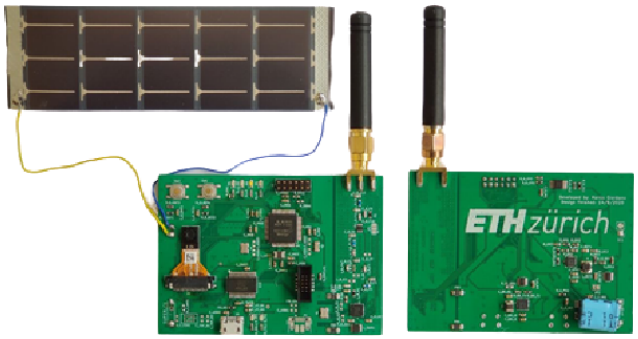

Figure 8. Face detection sensor system based on IoT [30].

2.7.4. Anomaly Detection

The difference between the mass of the majority of points and the point value is the anomaly. In [132], a study is given to see if TinyML is appropriate for tasks connected to anomaly detection. The Arduino Nano 33 BLE sense unit makes use of a general artificial neural network, an automatic encoder, and a variable automatic encoder. To check for irregularities in the uneven spin drying cycle, the study makes use of a Kenmore top-load washing machine type. The findings reveal that the precision is 90% and the accuracy is 92%.

2.7.5. Phenomenal and Ecological Maintenance

The study of phenotypic change at the gene level during the lifetime of an organism is known as phenomics. Particularly in plant phenomics, suitable shoots are chosen from a set of shoots via genetic changes. In [133] illustrates a phenomics image analysis based on the categorization of tomato leaf disease and spider mites. The YOLO3 method (based on the DarkNet-53 architecture) and the tomato PlantVillage dataset are used to automatically identify tomato leaves. The study also segments images into pixels using the SegNet method. In order to facilitate the application of phenomics under the banner of the TinyML paradigm, it also explores further into a number of other frequently employed methods of data analysis.

Recently, there has been an increase in the use of AI-aware tools for ecological maintenance analyses. In [134] develops an ecological conservation project in the SmallSats scenario using TinyML (e.g., small payload satellites). Unplanned fishing and marine pollution are causing the extinction of sea turtles. Using computer vision, sea turtle conservation has recently been integrated to the TinyML framework. The study focuses on increasing sea turtle conservation through the use of contemporary TinyML systems that are based on real-time vision.

Another important application that may be incorporated with TinyML systems is environmental monitoring. An approach to numerical weather prediction (NWP) using a deep tiny neural network (DTNN) is presented in [38]. The X-CUBE-AI tool chain, STM32 MCU, and the framework are all running the Miosix operating system. The sole hardware requirements for the DTNN architecture are 480 Bytes of on-board RAM and 45.5 KB of

flash memory. The weather forecasting system powered by a Raspberry Pi is seen in action in Figure 9.

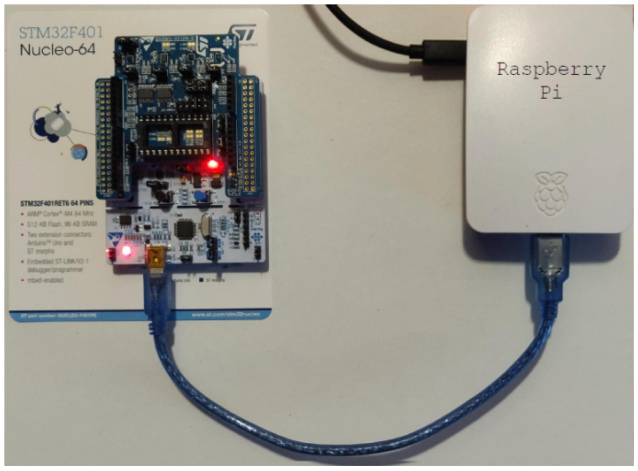

Figure 9. Weather forecast with STM32 microcontroller and Raspberry Pi [30].

2.7.6. Autonomous Vehicle and Traffic Management

However, it is presently a challenging task to drive a little automobile autonomously on such a small scale. The TinyML method is being researched to improve the autonomy of such tiny cars, according to [135]. Convolution neural network architecture is used in its construction in MCI GAP8. Additionally, the STM32L4 and NXP k64f platforms have been used for testing. The integration increases power consumption by 92% and reduces processing delay by 13x, according to the results.

Recently, automated traffic programming has been investigated for prospective integration with the TinyML framework in order to improve the real-time traffic management system [52]. Piezoelectric sensors are used in the system indicated, and they are placed in various traffic lanes. To identify a vehicle using piezo sensor data, a two-point time ratio approach is needed. In order to forecast when the green light will emerge, the categorization of cars is then employed. The random forest regressor (RFR) of the study considers the number of cars entering each lane when predicting signal duration. It is built on an Arduino Uno and uses scikit-learn's help along with the m2gen library. Only 1754 KB are needed for the created algorithm.

2.7.7. Body Pose Evaluation

Estimating body posture can be quite useful for keeping track of older patients health care. A platform-independent framework is added in [136], enabling quick model-platform modifications and validation. It creates the application on the Nvidia Jetson NX platform utilizing the face-landmarking technique RetinaFace and the posture estimation algorithm OpenPifPaf, which makes use of Composite Fields for real-time spatiotemporal posture detection (consisting of GPUs, deep learning hardware accelerators). Scalable inference for machine learning models is possible thanks to the assumed model-to-approve (MATE) checking framework (see Figure 10). It also offers image transformation, allowing the same model to be created on several platforms. Kestrel is a use case that tracks the body posture of the user at home or in the hospital. An alert is triggered whenever the user moves out of their safe place. MATE includes the TensorRT library to make it easier to incorporate the inference engine at different access levels.

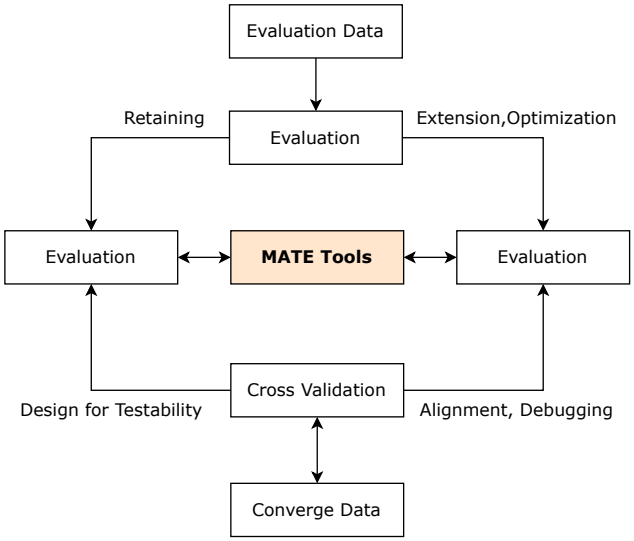

Figure 10. The MATE framework.

2.7.8. Detection of Respiratory Symptoms Associated with Coughing

Tiny RespNet, a convolution model with neural network awareness, is described in [62]. The approach is flexible and appropriate for multimodal settings. The system was constructed using a parallel processing-capable Xilinx Artix-7 100 t FPGA. The energy efficiency is 4.3 times greater than alternatives and the power consumption is 245 mW. The NVIDIA Jetson TX2 SoC was used to build the model, which is also evaluated against the processing power of the TX2 CPU. The input data may be categorized by the Tiny RespNet system, including voice recordings, demographic information, and patient records. Three datasets—ESC-50, FSDKaggle2018, and CoughVid—are used to categorize respiratory symptoms associated with cough detection (see Figure 11). Dyspnea can be recognized by the framework. Another project that uses the Arduino Nano BLE Sense platform with Edge Impulse cloud integration and is accessible on ProjectHub on the Arduino website shows off a cough detection application [137].

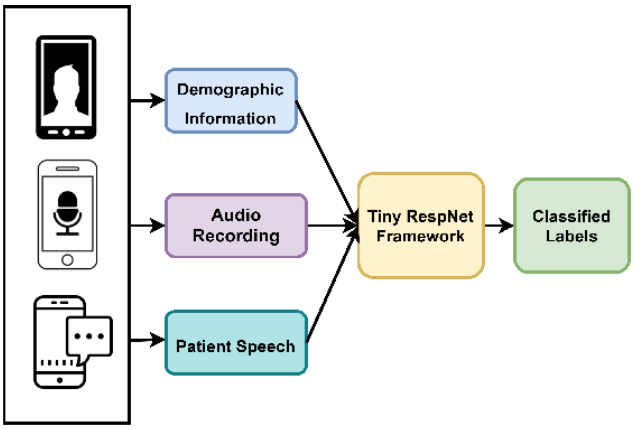

Figure 11. The Tiny RespNet.

2.7.9. Speech-Voice Recognition

Any machine learning model can employ speech recognition as a generic application. One such framework is TinySpeech, which can carry out on-device low precision speech recognition [74]. To benefit from small-footprint memory use and low CPU energy consumption, TinySpeech uses attention condensers. For the purpose of evaluating this efficacy of the framework, Google Speech Commands is used as a benchmark. In terms of parameters, it is discovered to drastically reduce architectural complexity by 507x. It can

do 48 times less multiplication and addition operations. Memory requirements are substantially reduced, 2028x lower than in earlier reported work. With a restricted vocabulary, it can identify voice recognition. The framework accepts the cephalic frequency coefficient (MFCC) representation of an audio signal as input. A softmax layer, an averaging layer, a fully linked layer and a convolution layer are all included. From an architectural standpoint, it has a lot of variety. It employs awareness condensers and a sparse application of generalized autonomous convolution. Furthermore, no batch normalization is applied. In this paper [138], they demonstrated efficient operation using both hardware and software to perform a TinyML audio classification on a resource-constrained embedded system. The experimental results showed that the time and power consumption were reduced when performing the operations on the embedded board. The proposed hardware-based preprocessing technique is suitable and can be applied to TinyML applications that require a large amount of raw information preprocessing in addition to the audio classification. As a final point, among the whole TinyML framework flow, software and hardware are separated based on the alignment function. The optimized partitioning between hardware and software co-design may provide a new topic for future research.

2.7.10. Oral Tongue Lesions Pre-Screening

A great way to reduce patient mortality is by early identification of oral cavity cancer (OCC). However, the pre-screening procedures now available are manual, and the therapeutic counseling that follows is not economically viable for the average person, especially in developing nations. They provide an automated, low-cost pre-screening technique in this work [139] that leverages Artificial Intelligence (AI) used in implanted acne equipment to recognize benign and pre-malignant superficial lesions of the oral tongue. The proposed machine learning technique retrains a MobileNetV2 neural network via transfer learning utilizing clinically annotated image datasets of nine different kinds of oral tongue surface lesions. For application on an embedded OpenMV Cam H7 Plus endpoint device with limited power and resources, they quantized a 32-bit floating-point precision floating-point model (float32) to an 8-bit integer model (int8) using TensorFlow Lite for microcontrollers. The quantized int8 model has a 98.69% accuracy in detecting 9 linguistic mistakes in the test set. Comparing the int8 model to the float32 model, which had the same performance at the same relative inference speed (1.1 ms), the int8 model used around 60% less RAM and flash memory.

3. TensorFlow Lite—TensorFlow Lite for Micro

3.1. Overview

The absence of appropriate frameworks has slowed the acceptance and deployment of TinyML in a range of products. The framework must provide not only the installation of a model on an embedded device but also the training of the model on a more advanced computer platform. TinyML must make use of a wide range of machine learning (ML) capabilities, as well as models organization and debugging tools that are helpful for production devices.

This gap has previously been attempted to be filled. We can analyze the main challenges that the framework is facing in the following [66]:

- Inability to install models in many embedded architectures easily and portably;
- There is an absence of optimizations that make use of the hardware in question without needing the construction of frameworks to execute platform-specific efforts;
- Lack of productivity tools that link training with platforms and development tools;
- Infrastructure for model calling, quantization, compression, and execution is insufficient;
- Minimum support features for debugging, organization, performance profiling, etc;
- No benchmarks exist that enable manufacturers to precisely and repeatedly measure the performance of their semiconductor;
- There is a lack of testing in real-world applications.

TensorFlow Lite Micro (TFLM) (URL: https://www.tensorflow.org/lite, tensorflow.org/lite) is suggested, to overcome these concerns, which emphasizes portability and flexibility while slowing down and increasing the cost of training and model deployment on embedded hardware. TFLM provides TinyML application execution on many architectures and enables hardware suppliers to gradually tune cores for their devices. The following advantages are provided to suppliers as well as a neutral platform to showcase their performance:

- The compiler-based solution is versatile, portable, and simple to include new applications and features.
- To achieve hardware independence, decrease the use of library requests and external dependencies.
- It allows hardware suppliers to provide kernel-specific optimization platforms without the need to develop hardware-specific compilers.
- Benchmarks adopted by leading benchmarking organizations such as MLPerf are provided.
- The framework is compatible with popular, well-maintained Google apps under development.
- It allows hardware suppliers to easily incorporate optimizations into their kernel to guarantee production performance and hardware benchmarking.
- The TensorFlow Lite model transformation and optimization infrastructure is one of many machine learning ecosystems that the model architecture framework is compatible with.

Proceeding with our overview of TensorFlow Lite, a brief reference to the work flow of the framework is needed. As shown in Figure 12, the neural network training process demands high computational hardware. So, it is trained on the generic TensorFlow model. Furthermore, training is only necessary if a customized dataset is suitable for a deep learning model, as applications can also utilize the pre-trained models of the framework.

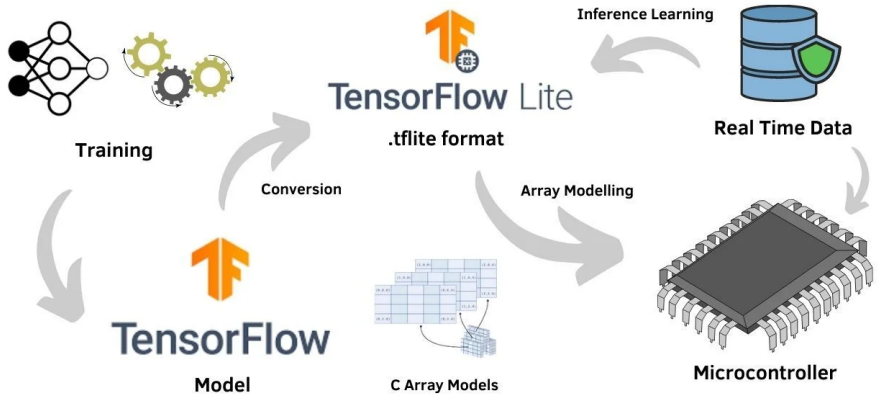

Figure 12. Workflow of TensorFlow Lite [66].

Considering a bespoke use case with a custom dataset, the user trains the model using the generic TensorFlow framework with high processing capacity and architecture. After training is completed, the model is evaluated using test procedures to confirm its accuracy and dependability. Furthermore, the TensorFlow model is converted into a hardware compliant TensorFlow Lite model in .tflite format.

The .tflite format is a buffer layer file used by the TensorFlow Lite framework and associated hardware. The model may then be used to train conclusions on real-time data received by the model. Models are optimized for resilient use cases through inference training. As a result, for edge AI applications, the choice of inference training is essential.

Most microcontroller firmwares do not support the native file system for the direct integrated flat buffer format of the TensorFlow Lite model. As a result, it is critical to convert the .tflite file to an array structure format suitable with microcontrollers. A simple way for this conversion is to include the program in the C table, followed by regular

compilation. The output format serves as a source file and consists of a character array suitable with microcontrollers.

TensorFlow Lite is appropriate for powerful devices, although it has a higher CPU overhead. Although the tiny size files in TensorFlow Lite Micro are prone to partial adaptation, optimizing the file size to fit in the memory can greatly increase performance for low power and processing devices such as microcontrollers.

The official TensorFlow documentation lists the following development boards that support the TensorFlow Lite Micro [140]:

- Adafruit Circuit Playground Bluefruit
- Kit Discovery STM32F746
- Adafruit EdgeBadge
- Espressif ESP32-DevKitC
- Himax WE-I Plus EVB Endpoint AI Development Board
- Wio Terminal: ATSAMD51
- Kit Adafruit TensorFlow Lite for microcontrollers
- Espressif ESP-EYE
- SparkFun Edge
- Arduino Nano 33 BLE Sense

TensorFlow Lite Micro is now available as an Arduino library for further microcontroller compatibility. For hardware development environments like Mbed, it could also produce projects.

3.2. Technical Challenges

Several factors make establishing a robust ML framework for embedded microprocessors especially challenging; some technical challenges are discussed below:

- Missing features:

Embedded all-in-one systems are distinguished due to their severe constraints. As a result, many of the current advancements that have sped up and simplified software development are not available on these platforms due to resource trade-offs being too expensive. Among the technologies [20] are an operating system, a standardized instruction language, floating point hardware, efficient memory management for dynamic provision, virtual memory, a file system, and other technologies that look simple to present programmers. A framework aiming for wide market adoption should not rely on specific platforms, even though they do offer some of these qualities.

- A decentralized market and environment:

Numerous applications for embedded systems demand solid software to be developed concurrently with the hardware, often by a collaborative team. Because of this, the lack of application compatibility of the platform is far less important than it is with ordinary-purpose computers. Reverse compatibility with outdated software is even less important on embedded systems than on main systems as nearly anything that runs on an embedded system is constructed from some fundamental source code in any case. Hence, even the most advanced x86 CPU system can still carry out almost thirty-year-old instructions, and embedded technology may be substantially changed to satisfy power demands [141].

These challenges comprise it far less likely for the embedded market to converge across one or two architecture platforms or ISAs, which will cause fragmentation. In many ISAs with robust ecosystems, the advantages they offer for certain applications exceed the cost of switching developers. Businesses still let developers create their own ISA extensions nowadays [142].

The number of embedded architecture-supporting tool chains and integrated development environments (IDEs) is proportionate to the number of embedded architectures. Most of these systems can only be accessed with a business license from the hardware vendor, and if a client requests certain pre-defined instructions, they might not be available

to all customers. These setups lack an open source environment, which leads to device fragmentation and makes it impossible for a single development team to provide software that is effective across a range of embedded systems.

- Resources are limited:

Embedded hardware developers do this because the capabilities of a general purpose computing platform are beyond them. The most important factors are power consumption (in embedded devices, there is the requirement for a few milliwatts in terms of power, while conversely mobile phone and desktop processors require watts [143]), cost (a microcontroller typically sells for less than a few dollars [144]), and form factor (efficient microcontrollers can be as small as a grain of rice [143]). To meet the demands of consumers, hardware developers exchange their skills so as to collaborate and communicate ideas. In most cases, the embedded system built has a small amount of memory, however this do not apply as a general rule. On the other hand, a big embedded system with little over a megabyte of SRAM and several megabytes of flash memory is one extreme paradigm within the spectrum of all-in-one devices. A tiny embedded system, however, contains processing memory that is often shared between RAM and ROM and has a memory capacity of a few hundred kilobytes or less [11].

Persistent storage and working memory are significantly less than typical software created for general-purpose computers as a result of these limitations. The amount of the compiled code in storage, in particular, must be decreased.

Code that is rarely utilized on a single device can be found in the majority of applications made for general-purpose platforms. Instead of choosing the route of the code at runtime, it would be more efficient to send more executables that have been specifically created. Since code size is a concern and there are few practical uses, it might be difficult to defend such runtime flexibility. Developers must thus dissect a library abstraction in order to make changes that are appropriate for their intended hardware.

3.3. Implementation of TensorFlow Lite—TensorFlow Lite Micro

In this section we will try to explain in detail the implementation of TensorFlow Lite for Micro, as shown in Figure 13 below, in order to have a schematic view of the Google framework based on [66].

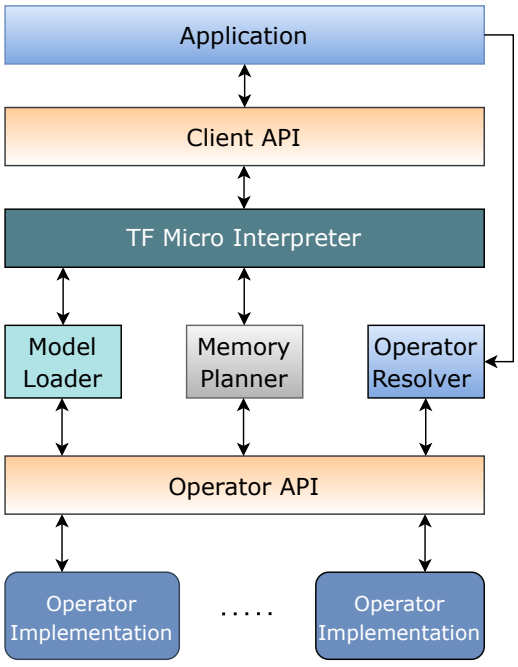

Figure 13. Implementation overview TFLM.

- ***System overview***

The initial stage of a TFLM development process for application creation is to memorize an active neural network model. Using the client API, the developer of the application creates an "operator resolver" object. To reduce the file size, the "OpResolver" API regulates which operators are associated with the final binary. The second stage is to supply the interpreter with a continuous "region" of memory holding the intermediate results and other variables. Because we presume that dynamic memory allocation is not accessible, this is necessary action that has to be performed. The final step is to develop an interpreter instance and deliver the model onto it along with a resolution operator, and a given region as parameters. The interpreter allocates all necessary memory from the arena during the initialization phase. Any further allocation can be forged to ensure heap fragmentation and to avoid creating issues that frequently appear in long-running apps.

Operator programs can allocate memory for use during evaluation, allowing the interpreter to express memory demands. The operator types of the serialized model are mapped to the implementation functions by the OpResolver provided by the implementation. To guarantee that operator implementations are independent of interpreter details and modular, all communication between the interpreter and the operators is handled via a C API call. This approach encourages the reuse of existing system implementations while enabling the replacement of operator implementations with optimized alternatives (e.g., as part of a code generation project). Implementing the plan is the fourth phase. The software gets pointers to memory spaces that represent the model inputs and fills them (often derived from sensors or other user-supplied data). Once the inputs are ready, the application calls the interpreter to compute the model. In this process, topologically ordered operations are iterated through, inputs and outputs are identified using offsets discovered during memory scheduling, and the evaluation function is executed for each operation.

The translator reviews all actions before handing over command to the program. It is just a simple blocking call. Because most MCUs are single-threaded and depend on interrupts for important operations, this is acceptable. However, platform-specific operators can continue to split their work amongst several processors, and a program can continue to run on a single thread. After the call is complete, the application can instruct the interpreter to find and use the tables containing the calculation outputs of the model. Since threading and multitasking would require less operating system requirements and portable code, the framework does not support them. It does, however, permit a range of uses. As long as they do not required to run simultaneously, the framework may execute many models [66].

- ***TFLM Interpreter***

TFLM is a machine learning framework for structural inference that is interpreter-based. A data structure that directly expresses a machine learning model is loaded by the interpreter. The interpreter manipulates the data of the model during runtime, which defines the operators needed to be executed and where the arguments of the model are drawn from, despite the fact that the execution code is static. We select an interpreter based on our expertise designing production use models in embedded hardware. We recognize the requirement for models to be easily updated in the field, which may be impossible with production code. Nevertheless, utilizing an interpreter enables greater code sharing between models and applications and speeds up code maintenance because it enables changes without re-exporting the model. The long-term complexity of the kernel also favors ML model interpretation, in contrast to conventional interpreters that have several branches in connection to a function call. Because each kernel runtime is lengthy, interpreter overhead is decreased.

By rendering operator function calls in fixed machine code at export, a model can generate C or C++ source code as an alternative to an interpreter-based inference engine. The code must be recompiled for each model, which might improve performance but decrease portability. Code generation, which interpolates parameters like weights, model architecture, and layer dimensions into the binary, is used to completely replace an exe-

cutable while editing a model. Since an interpreter method maintains all of these data in a distinct file or memory region, a model update can substitute it for a single file or continuous area of memory.

- ***Loading of the model***

As was already mentioned, the interpreter loads a data structure that specifically represents a model. TensorFlow Lite portable data schema [145] is used for this study. We may import a large range of models at minimum technical expense by reusing the extraction methods from TensorFlow Lite.

– *Model serialization*

The FlatBuffer serialization standard is used by TensorFlow Lite for mobile devices to retain models. Less than two kilobytes of binary space are generally used by the access code. Additionally, a header-only library exists that reduces space and facilitates compilation by eliminating the requirement to decompress the serialization protocol into another format.

The C++ header demands that the platform compiler implement the C++11 specification, which is a downside of this format. The fact that most embedded devices lacked file storage systems was another obstacle for this format; nevertheless, because it uses a memory mapped representation, it is straightforward to transform the files into C source code files that include data tables. All of these files have been combined into binary files that the application can easily access.

– *Model representation*

The model is represented by the stored data, the value schema, and the TensorFlow Lite representation. This schema offers a number of characteristics that make programming for embedded devices simpler because it was created with storage effectiveness and quick access for mobile systems in mind. In contrast to a directed acyclic graph, the functions are retained in a topologically ordered list. A comprehensive representation of the graph would need preprocessing to match the input dependencies of the operation, but computations are as simple as iterating over the list of operations in order. The drawback in this representation is that it is designed to be portable between systems, necessitating runtime processing to provide the data needed for inference. As an illustration, it takes the operator parameters out of the inputs before passing them to the functions that carry out those operations. Because of this, each action requires a little piece of runtime code to convert from the serial representation to the underlying structure of the implementation. The readability and compactness of the operator implementations degrade even if there is no code cost. Programming in memory is a comparable problem. Any reliant operations may see size variations since TensorFlow Lite for mobile devices enables variable size inputs. After all of the buffer dimensions are determined, the optimum arrangement of the intermediate buffers for the computations must be decided at runtime.

- ***Memory management***

It is not possible to assume that the operating system is capable of allocating memory in a dynamic manner. As a result, the framework uses an accessible memory space for memory allocation and maintenance. When the interpreter initializes the model, it calculates the total size of all buffers and lifespan needed to execute the model. Runtime tensors, permanent memory for data storage, and scratch memory for temporarily storing values during model execution are all contained in these buffers (discussed below). The framework creates a memory plan after determining all necessary buffers, using non-persistent buffers whenever feasible and ensuring that the buffers are still valid at the conclusion of the lifespan of the memory plan.

– *Persistent Memory and Scratchpads*

Applications must supply a memory field with a specified size when the interpreter is formed and must keep this memory field constant in the lifespan of the interpreter. This region may be treated as a stack by assignments that have the same lifespan. We produce an application-level error if an assignment uses up too much space.

We ensure that allocations only take place during the setup phase of the interpreter, preventing memory faults from interfering with a long program. The model call has no possibility of allocation (through mechanisms).

This simpler technique is useful for initial development, but it wastes memory because several assignments may overlap in time with others. Data structures that are only required during startup are an example. Their values are meaningless after initialization, but because they have the same lifespan as the interpreter, they continue to occupy space in its region. Variables that do not need to remain constant between calls are also necessary for the assessment phase of the model.

In order to store the initialization and evaluation lifetime distributions independently from the interpreter lifetime objects, we modify the distribution system. For the interpreter lifetime assignments ("Tail" in Figure 14) and the function lifetime objects ("Head" in Figure 14), this approach employs stacks that rise from the lowest direction and decrease from the highest direction, respectively. A limitation of capacity is indicated when the two stack indices cross [66].

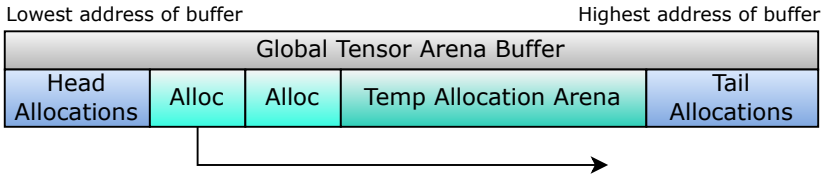

Figure 14. Dual-stack allocation technique.

Both shared and permanent buffers benefit from the two-stack allocation technique. However, model initialization retains allocation data that is no longer required for model inference. As a result, while a model is memory planning, the space between the two stacks might be used as a temporary allocation. The permanent assignment stack, a component of the permanent stack, maintains any temporary data needed for model inference.

- ***Multi-tenancy***

Due to embedded system limits, application model developers may be forced to construct multiple specialized models rather of a huge single model. As a result, support for many models may be required in the same embedded system. It is feasible to have two different instances that operate independently of each other if an application contains various models that do not need to execute concurrently. However, because the cache cannot be reused, this is inefficient.

On the other hand, TFLM offers multi-tenancy along with certain transparent to the programmer changes to memory scheduling. Memory region reuse is made possible by TFLM, which enables several model interpreters to commit memory from a single area. It permits reuse of the lifetime section of the functions for model assessment and stacking of the lifetime regions of the interpreters in the region. Based on all of the models assigned to the arena, the reusable (non-permanent) component is set to the strictest standard. The non-reusable (permanent) allocations for each model are raised.; this is known as model-adjusted allocations [66].

- ***Multithreading***

As long as the memory allocation of the model is contained inside the area and no model-related information is kept outside the interpreter, TFLM is thread-safe. In the

region, just the interpreter variables are kept, and each interpreter instance is associated with a particular model. Because each interpreter instance is handled by a different task or thread, TFLM is capable of managing a large number of interpreter instances. Even on several MCU cores, TFLM may operate safely. Since only the variables required by the interpreter are kept in the domain, this really functions nicely. The regions make sure that there are no threading problems even when the executable code is shared.

- ***Operator Support***

They need a lot of computation, occasionally requiring hundreds or even billions of sequential arithmetic computations (e.g., addition or smultiplications). They operate smoothly, include inputs/outputs, and have state variables that are specified. Moreover, they seem to have no additional side effects.

These processes are frequently customized for specific platforms in order to take use of hardware characteristics since power consumption, code size, and model execution latency dominate implementations of these techniques.

It is possible to create an API that connects inputs and outputs while hiding execution specifics behind an abstraction since operator bounds are clearly specified. Some chip manufacturers have made a library of neural network kernels accessible in order to enhance the performance of neural networks when run on their CPUs. The fully connected layer, softmax, convolution, aggregation, activation, and optimized fundamental math are just a few examples of the optimized CMSIS-NN libraries that Arm provides. CMSIS-NN is used by TFLM to get improved performance [66].

- ***Build all-in-one System***

To address the fragmentation of the embedded systems market, we need our code to be compilable across many different platforms. Therefore, we create the code to be mostly portable, with fewer dependencies, but not enough to provide a decent experience on a certain device. The majority of embedded system programmers work with a platform-specific integrated development environment (IDE) or tool chain, which abstracts away many of the intricacies of building individual components and offers libraries as interface modules. Even if developers were just given a hierarchy of folders holding source code files, before they could create and compile that code into a useful library, they would still need to go through a number of processes.

3.4. Summary of TensorFlow Lite

First, special reference is made to TensorFlow Lite because this framework is widely used, can solve many problems for TinyML technology and is supported by Google. In the previous chapter, other frameworks and compilers for TinyML have been mentioned, such as the ELL of the Microsoft, STM32Cube.AI etc. The purpose of this chapter is to give a brief overview of the technology of TensorFlow Lite for Micro as well as to explain some complex issues related to the implementation of this framework. Figure 15 below shows the pipeline of this framework based on [146].

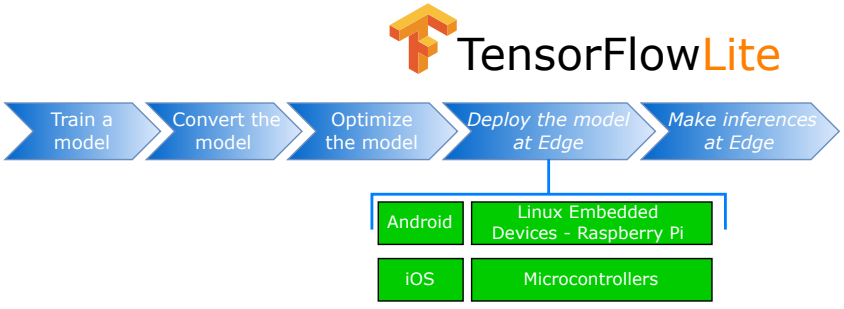

Figure 15. Standard pipeline for TensorFlow Lite.

The capacity to transfer deep learning to embedded devices is made possible by TFLM, greatly expanding the use of ML. The TFLM framework was developed particularly to do machine learning on embedded systems with low memory—only just few kilobytes, in fact.

The TensorFlow Lite deep learning framework expands the opportunities in a number of AI applications. Because the framework is open source for AI enthusiasts, it is becoming increasingly popular for machine learning use cases. TensorFlow Lite as a whole improves the environment for developing edge applications for embedded and IoT devices.

There are also various examples for newcomers [80] to assist them with real use cases of the framework. Face identification based on data obtained from the development board's image sensor is one of these instances, as is the typical hello world application for all development boards. Examples can include motion detection and speech recognition apps for specialized development boards.

4. Integrating TinyML with Network Technologies

4.1. 5G and TinyML

At this point, we'd like to talk about another technical breakthrough that will transform how devices communicate with one another: 5G. With minimal latency, 5G connectivity can play a special role in managing huge amounts of data [147,148]. 5G will usher in a new hyper-connected society in which everyone and everything will be linked. The relationship between the ML, the 5G, and the TinyML is depicted in Figure 16.

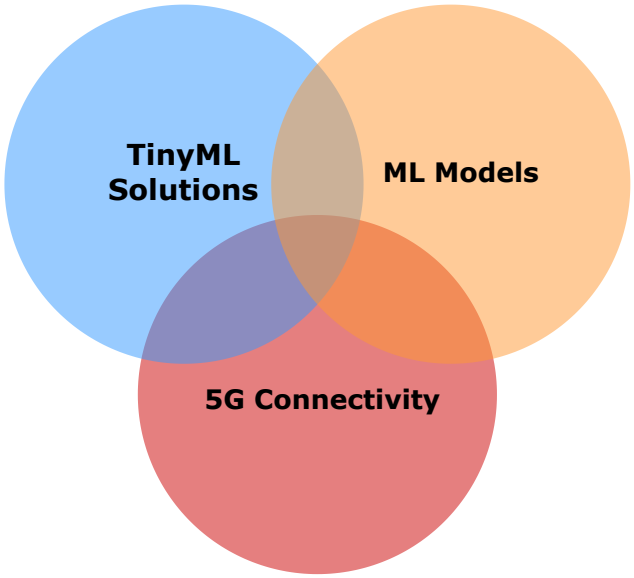

Figure 16. Association of ML, 5G and TinyML.

As a result, the coming of 5G will usher in new paths and challenges for TinyML. We would want to suggest that, while TinyML is meant to handle ML challenges at the edge, predictions based on metadata or collective decisions made by thousands of devices will continue to be made in the cloud. As a result, it would be advantageous to distinguish between issues solved by devices and problems solved in the cloud. Furthermore, incorporating the benefits of both technologies while avoiding their drawbacks would result in a more resilient design. Researchers think that 5G will play an important role in more successfully linking these two frameworks because of the following vision [149].

- The real-time data rate is targeted at =1~10 Gbps [150], with a delay <10 ms [151].
- A significant number of devices are linked together [152].
- The goal is to minimize energy use by about 90% [147].
- It offers massive bandwidth, cheap cost, and extended battery life.

In light of the development of machine learning and its benefits in wireless communications, each scientific community has made an attempt to assess the influence of ML

on 5G in their individual domains. As a result, multiple papers have been written about various themes like the effect of machine learning on the physical layer, privacy concerns, managing radio resource, and more. This makes it difficult to give a brief explanation of the importance and use of ML/AI in 5G. Due to the high complexity of 5G, deep learning has been identified by many leading papers as the most promising ML method. As a result, works that apply ML to 5G can be divided into two categories: "deep learning-based classification", in which we place a sole focus on deep learning and "general ML classification", where we take into account all feasible ML viewpoints from the literary works [153].

In the case of 5G, a broad ML classification follows the general ML structure shown in the previous sections, which employs three ML categories: Unsupervised learning, supervised learning and reinforcement learning (RL). An illustration of such categorization is provided in Table 6, providing the learning methodologies utilized in each category as well as a specific example of a 5G [154] application. Moreover, RL and 5G can exist in unmanned aerial vehicles (UAVs) as indicated in [155]. The next part will outline several 5G use cases and provide a method for mobile network operators to integrate AI/ML.

Table 6. Some Learning Methods and their Applications in 5G for the Three Types of Machine Learning.

Learning Classes	Learning Models	5G Application Illustration
Supervised learning	Support Vector Machines (SVM)	Model for predicting path loss in urban contexts
	Approaches for machine learning and statistical logistic regression	In deployments of self-organized LTE dense small cells, dynamic frequency and bandwidth allocation is used.
	Neural-Network-based approximation	Channel learning is used to infer unobservable channel state information (CSI) from an observable channel.
	Frameworks for Supervised Machine Learning	Adjustment of the TDD Uplink-Downlink configuration in XG-PON-LTE Systems to enhance network performance in the hybrid optical-wireless network based on current traffic circumstances
	Multi-Layer Perceptrons (MLPs) and Artificial Neural Networks (ANN)	In next-generation wireless networks, objective function modeling and estimates for link budget and propagation loss are used
Reinforcement Learning	Reinforcement Learning algorithm based on long short-term memory (RL-LSTM) cells.	Based on long-term WLAN activity in the channels and LTE-U traffic loads, proactive resource allocation in LTE-U networks, implemented as a non-cooperative game, enables SBSs to determine which unlicensed channel to utilize.
	the modified Roth-Erev (MRE), Gradient follower (GF), and the modified Bush and Mosteller (MBM).	Allow Femto-Cells (FCs) to monitor the radio environment autonomously and opportunistically and alter their settings in HetNets to eliminate intra/inter-tier interference
	Reinforcement Learning with Network assisted feedback.	Selection of Heterogeneous Radio Access Technologies (RATs)
Unsupervised Learning	Clustering using Affinity Propagation.	Data-Driven Resource Management for Ultra-Dense Small Cells
	Hierarchical Clustering.	Detection of anomalies, faults, and intrusions in mobile wireless networks
	ML Framework for Unsupervised Soft-Clustering.	In heterogeneous cellular networks, latency is reduced by grouping fog nodes to automatically identify which low power node (LPN) gets converted to a high power node (HPN)
	Expectation Maximization (EM), K-means clustering and Gaussian Mixture Model (GMM).	Relay node selection and cooperative spectrum sensing in vehicle networks

Deep learning is the primary AI/ML learning technique that certain studies concentrate on since it is thought to be the most effective. As described in earlier case studies [156,157], this gives 5G and its numerous components a very wide variety of applications. A number of studies have also concentrated on the use of several very promising deep learning subvariants, for instance traffic engineering/resource planning, network security and connection preservation, deep Q-learning for caching/offloading [158]. The advantages of using deep learning and TinyML to handle wireless communications issues, particularly in the context of 5G, are outlined in [156] and may be summed up as follows:

- Unsupervised learning: Unsupervised learning is achievable due to the efficiency of deep learning in processing semi-labeled or unlabeled input. This is essential for dealing with the enormous amounts of unlabeled data that mobile systems frequently deal with.

- In contrast to conventional ML methods, deep learning performance usually increases rapidly as the size of the training data increases. As a result, it can exploit the enormous amounts of mobile data created at high rates of speed.
- Geometric mobile data learning: Geometric mobile data analysis has been transformed by specific deep learning architectures for modeling geometric mobile data.
- Extraction of features: Through layers of varied depths, deep neural networks can automatically retrieve high-level information. This makes it possible to analyze heterogeneous and noisy mobile big data with lower cost human feature engineering.
- Multitask learning: Using transfer learning, features gained from neural networks via hidden layers may be applied to other tasks. This minimizes the processing and memory needs for doing multitask learning in mobile devices.

4.2. LPWAN and TinyML on Embedded Devices

Low power wide area networks (LPWANs) aim to reduce energy consumption but at the same time to increase distance coverage. As indicated in Figure 17, LPWAN networks tend to have low data rate ratio vs. power consumption but sufficiently high range while in some cases the range exceeds more than 10 kms.

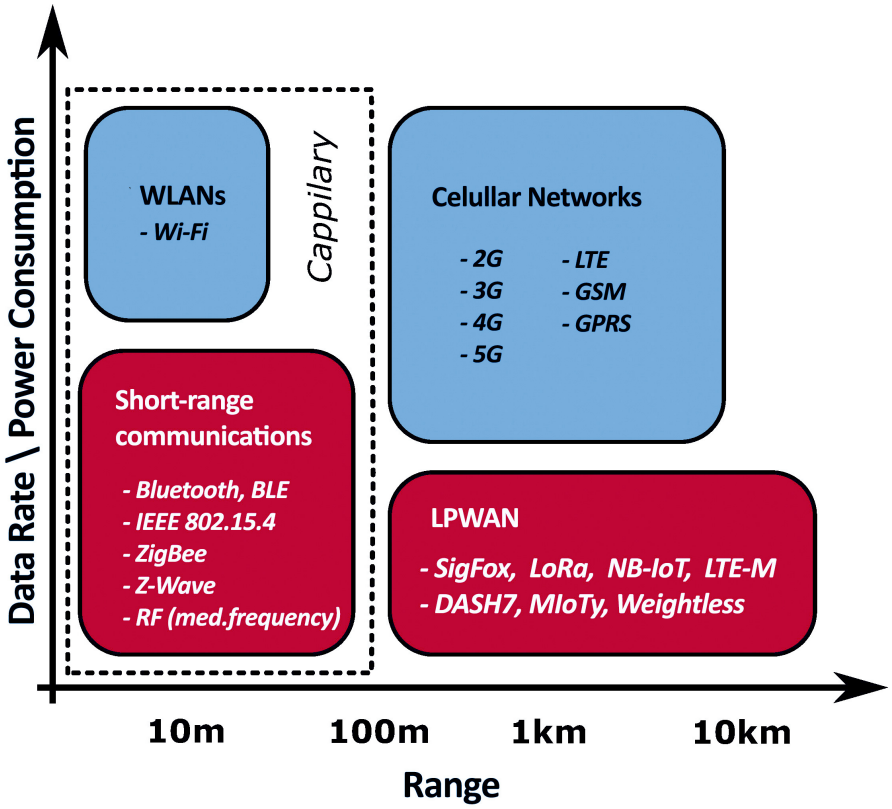

Figure 17. Comparison of network protocols.

TinyML is an emerging paradigm that efficient ML algorithms can be deployed on wearable devices [159]. However, utilizing such networks in a federated learning (FL) scenario can lead to the creation of an ML model from decentralized data and in a distributed way. This is the fundamental concept of FL where a variety of edge devices collaborate so as to build a global model using only local copies of the data and then each device downloads a copy of the model and updates the local parameters [160]. Finally, the central server aggregates all model updates and proceeds with the training and evaluation without exchanging data to other parties. Table 7 depicts the comparison of LPWAN technologies as per various parameters. Furthermore, based on the literature of modern network technologies and embedded systems we present the scalability, deployment, cost efficiency,

battery life, security and quality of service based on the network technology that is being utilized. In Table 8, these traits are further shown.

Table 7. Comparison of LPWAN technologies (Compiled from [161–167]).

Parameter	LoRa	SigFox	NB-IoT	LTE-M	DASH7
Standard	LoRa Alliance	SigFox/ETSI LTN	3 GPP Release 13, 14	3 GPP	Dash Alliance
Bandwidth	250 kHz	100 Hz	200 kHz	1.4–20 MHz	433/868/915 MHz
Modulation	FSS/CSS	D-BPSK	QPSK	DL: OFDMA, 16 QAM	GFSK
Spectrum	1175 kHz	200 kHz	200 kHz	Licensed LTE bands	Licensed
Frequency band	EU: 868 MHz	EU: 868 MHz	7–900 MHz	Cellular Band	Cellular Band
Transmission	FHSS (Aloha)	UNB	FDD	FDD/TDD	BLAST
Topology	Star-of-stars	Star	Star	Star	Half
Security	AES 128b	Optional encryption	NSA AES 256	AES 256	AES 128
Range (Urban)	2–5 km	3–10 km	1–5 km	1–5 km	1 km
Range (Rural)	20 km	50 km	10–15 km	10–15 km	2 km
Data Rate (Min)	250 bps	100 bps	100 kbps	1 Mbps	27.8 kbps
Data Rate (Max)	50 kbps	600 bps	200 kbps	4 Mbps	200 kbps
Throughput	50 kbps	-	127 Kbit	1 Mbit	167 Kbit
Energy Consumption	Very Low	Low	Medium Low	Medium	Low
Battery Life	∼10 years	∼12 years	∼10 years	∼2 years	∼10 years
Deployment Cost	Moderate	Moderate	High	High	Moderate
TinyML Availability	Yes	Not applicable	Not applicable	Yes	Yes

Table 8. TinyML applications vs. ordinary Machine Learning applications in network technologies.

Parameter	TinyML	Machine Learning
Battery Life	✓	
Cost efficiency	✓	
Scalability		✓
Robustness	✓	
Deployment		✓
Performance		✓
Security	✓	

As per the battery life, TinyML outperforms ordinary ML techniques as the models can run on embedded devices. Cost efficiency is better in TinyML as only one microcontroller is required compared to a PC. Scalability is higher in ordinary ML applications as more computing power is available. Robustness is higher on TinyML deployments as in the case when a node is removed, all information remains intact while on the ML case is server-side based. The deployment is better on ML models as there are many paradigms available online and more widely used. The performance metric is higher on ML cases as TinyML technology has emerged and there are not much models. Lastly, security is higher on TinyML deployments as the information remains within the embedded device and there are no exchanges between third parties.

5. Discussion and Future Directions

IoT devices will have flexibility and processing power never previously possible thanks to the incorporation of ML. TinyML is a resolution that performs best in low-resource settings because it provides a simple protocol, portability and intelligence that supports modern IoT devices. TinyML's data processing solution at the sensor node is essential for enhancing results despite the ongoing proliferation of IoT devices in cloud-based frameworks. However, new methods for ML algorithm optimization as well as hardware innovation for use in situations with limited resources have the potential to

change IoT architecture. While improving the ML algorithm, caution should be used as the performance of the categorizer may decrease. TinyML contributed to the creation of a new type of machine learning frameworks for the edge devices of the IoT ecosystem that are resource-constrained and battery-operated. The TinyML community continues to struggle with ML model optimization even as academics are focusing more on creating software frameworks for creating a scaled-down version of the ML model to embed in MCUs. TinyML use is rising across a number of industries. To promote innovation and ongoing investment in this industry, the effectiveness of the framework must be guaranteed. Consequently, TinyML needs a reliable benchmarking mechanism. We predict that a TinyML benchmarking solution that is precise, easy to use, rapid, and reliable enough to be utilized in a variety of IoT situations will soon be available based on our research to far.

Furthermore, TFLM makes it possible for deep learning to be applied to embedded devices, greatly expanding the use of ML. Designed for usage in embedded systems with memory capacities as little as a few kilobytes, a machine learning framework called TFLM was developed. The primary contributions of TFLM are the design choices that must be taken to address the unique challenges of embedded systems, such as the lack of software features, the severe resource limits, and the heterogeneity of hardware in a fragmented environment.

TinyML, in particular, has created a new class of machine learning frameworks aimed towards edge devices in the IoT ecosystem that run on batteries and have limited capabilities. Although academics have concentrated their efforts on developing software frameworks that try to provide a shortened version of the ML model for inclusion into MCUs, improving the model even so presents a demanding task in the TinyML society. In a variety of applications, TinyML use cases are occurring more often. As a result, the efficacy of the framework should be ensured in order to stimulate innovation and additional investment in this sector. As a result, TinyML requires an effective benchmarking mechanism. Based on our study thus far, we anticipate the creation of a TinyML benchmarking solution that is exact, simple, quick, and reliable enough to be applicable to a variety of IoT applications.

Finally, TinyML is encouraging the community to overcome issues that have never been addressed at such a level previously. Developers of TinyML must understand how data is used, how it is displayed, and what data is used. We could be asked to create unconnected, pre-installed systems with out-of-date software. A TinyML device may never have a network connection, which raises the bar even higher and means it cannot be the subject of an attack from an adversary or breach the privacy of a user. However, newer, more accurate alternatives can never take the place of such equipment. There are simply too many potential uses for compact sensors with integrated machine learning. The majority of these possibilities are in places without electricity, and even when power is available, additional constraints typically dictate reduced processing capability, near-zero cooling, limited communication capabilities and a tiny form factor. Last but not least, one of the most underestimated issues with TinyML is the incorporation of so-called "ethical AI" [37]. We must make sure that these products always adhere to what we, as humans, find ethical. As was previously said, rapid rise of TinyML has permitted products and systems that are now firmly ingrained in our culture. While aiming to use less resources, we must make sure that these systems were trained and designed without bias. Moreover, we analyze a future roadmap related to TinyML and based on 5 key components [30]:

- New dimension: A variety of variables are claimed to influence the development and acceptance of the TinyML standard in the future. An intelligent edge system, for example, necessitates edge software that integrates sophistication, edge-device intelligence consistency, real-time learning, distributed learning, online learning, and data-network management. This partnership should be expanded to strengthen local security and privacy by allowing end-user context to be stored on edge devices. Additionally, it should give priority to improving edge device infrastructure, low-cost knowledge sharing capabilities in edge device systems and edge device platform

orchestration. Figure 18 emphasizes the contributions of these factors to the evolution of the TinyML paradigm.

- Support for portability: Another issue that may be overcome by establishing improved compute data delivery methods on Edge platforms is portability support. Location-aware optimization can conserve network capacity and boost network spectrum coverage for edge devices, allowing them to collaborate with neighboring devices. It can therefore contribute to the progressive transfer of know-how and models for others to utilize. Furthermore, the quality of service may be altered to forecast how edge devices would behave when interacting with neighboring nodes. These data may be exchanged across nodes or clouds in the immediate vicinity to forecast the intelligent orientation of edge devices.
- Edge Intelligence Framework: The following should be the foundation of a standard edge intelligence framework: (i) energy-efficient management, (ii) dynamic task distribution, (iii) data intelligence, (iv) wireless networking, (v) collaborative intelligence, (vi) predictive service quality, (vii) communication service implementation, (viii) real-time inference, (ix) liquid software propagation between edge nodes, and (x) machine learning as a service. From a future-proof learning perspective, take into account implementing such TinyML arrangements to make them compatible with 5G and 6G technologies. Therefore, mmWave xhaul systems have to be included to TinyML systems to enhance ML models optimization. Hypercooperation between the cloud and the edge may thus be implemented in an effective manner [168–170].
- Offloading operations: The activation/deactivation of processing tasks can actually be exploited during machine learning scenarios triggered by the edge. These loading methods ought to be included to the dynamic configuration of edge-aware features that is already accessible. Thus, TinyML will enhance the transmission of resource-intensive processes from edge devices with low resources. Interoperability between the edge and the cloud could become more focused as a result. It is necessary to undertake investigations to determine the underlying processes that support these resource allocations (e.g., machine learning, channel bandwidth, memory chunks, data recruitment, sensing capabilities, CPU cycles).
- Future Perspective: TinyML is progressively becoming a must and a reality for making educated decisions in everyday life. Especially for low-power embedded devices used in a variety of applications. To provide consumers with an upgraded user experience, mobile platforms must adjust their orientation to TinyML. TinyML-aware approaches for next age intelligent and wearable devices are recommended for technical developers and enterprises. There is a great requirement to reduce CPU-GPU-TPU interaction, which costs resources, in order to provide smart decision support. Microcontroller makers should prioritize integrated TinyML design standards so that customers do not have to deal with external alignments linked to artificial intelligence. TinyML integration in the realm of IoT-edge analysis should be explored in order to make the application more user-friendly and trustworthy. To assist developers in implementing the market-ready development scenario, a uniform flow methodology should be designed. It is necessary to provide appropriate dataset repositories and lightweight benchmarking tools. TinyML adaptation should target X.0 industrial applications in the future days. Furthermore, such low-memory libraries should be used with 8-bit microcontroller devices. TinyML should be used to resolve latency mitigation in edge-level effects. TinyML is used in low-cost and portable digital devices to deliver immediate input to consumers. TinyML frameworks may be used to reduce needless utilization and reliance on GPUs, TPUs, and cloud platforms.

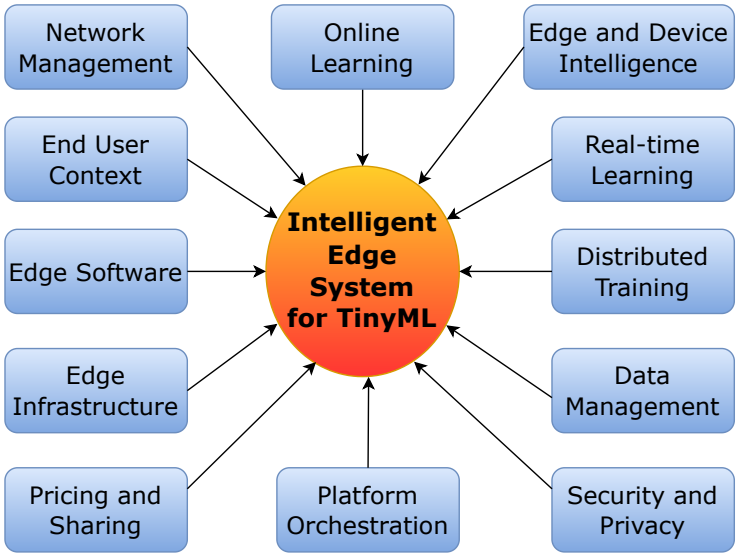

Figure 18. Basics key enablers of TinyML.

6. Conclusions

The Edge-IoT ecosystem has enormous potential [171,172] to exploit the ability to make smart decisions at the edge of the network. Machine learning engagement in small embedded devices with low resources is growing as a critical necessity, pushed by potential futuristic applications. To improve the present edge-driven machine learning framework, the TinyML paradigm should be further investigated. In this paper, we give the data on TinyML in depth, covering several aspects of the research. It is acknowledged that changing the TinyML paradigm with sufficient depth in in-memory processing and neuromorphic computing is likely possible. Before they may be applied to low power embedded devices, current benchmarks must be updated with new datasets. We anticipate that the evolution of the edge-IoT tool chain will require the use of a low-footprint machine learning output translator. By creating new TinyML process flows, benchmarks, optimized datasets and integrating TinyML into typical low-cost, low-power smart wearable devices like smartphones, microcontrollers, and IoT-Edge systems, we urge enterprises and stakeholders to take this step. TinyML has the ability to drastically alter how decisions are made, allowing for the efficient handling of complex problems even at the edge of the network.

To summarize, TinyML is a significant and quickly developing sector that requires hardware innovation comparability to enable stability and continuous advancement. In this journal, we examined the current situation of TinyML. In addition, we selected frameworks, libraries, and hardware platforms relevant to the TinyML industry. We reviewed the drawbacks of TinyML technology, its benefits, a number of use cases with specific examples, and its application domains with analysis for each field. We also investigated the connection of TinyML with 5G and LPWAN technology in embedded devices, with great potential for the future of this technology. We believe that this journal is a comprehensive guide that provides those interested in this field with a complete overview of this innovative technology. Consequently, the conclusions of this comprehensive analysis will be used as a guide for understanding the rapidly expanding field of TinyML, a foundation for future research in this field, and a valuable source for the scientific community. Last but not least, TinyML is a substantial and rapidly developing field that necessitates trade-offs between several essential components (hardware, software, machine learning algorithms). Future research might include other aspects, such as TinyML's application domain (smart agriculture and livestock, smart spaces, vehicle services, industry 4.0, e-health, etc.). TinyML has the prospect to unlock a plethora of new smart applications in industry, business, and private life. TinyML provides innovative solutions in a wide range of disciplines as well as new research pathways, making it an intriguing issue for scholars to examine. Aside from the development of new deep learning neural networks [68].

Author Contributions: N.S., A.K., C.K. and S.S. conceived of the idea, designed and constructed the review article, analyzed the applications of TinyML, drafted the initial manuscript and revised the final manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kuo, Y.H.; Kusiak, A. From data to big data in production research: The past and future trends. *Int. J. Prod. Res.* **2019**, *57*, 4828–4853. [CrossRef]
2. Ma, S.; Zhang, X.; Jia, C.; Zhao, Z.; Wang, S.; Wang, S. Image and video compression with neural networks: A review. *IEEE Trans. Circuits Syst. Video Technol.* **2019**, *30*, 1683–1698. [CrossRef]
3. Estrebou, C.A.; Fleming, M.; Saavedra, M.D.; Adra, F.; De Giusti, A.E. Lightweight Convolutional Neural Networks Framework for Really Small TinyML Devices. In Proceedings of the International Conference on Smart Technologies, Systems and Applications, Quito, Ecuador, 1–3 December 2021; Springer: Berlin/Heidelberg, Germany, 2022; pp. 3–16.
4. Carvalho, G.; Cabral, B.; Pereira, V.; Bernardino, J. Edge computing: Current trends, research challenges and future directions. *Computing* **2021**, *103*, 993–1023. [CrossRef]
5. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
6. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *arXiv* **2017**, arXiv:1707.01083.
7. MCUs-Sales-To-Reach-RecordHigh-Annual-Revenues-Through-2022. Available online: https://www.icinsights.com/news/bulletins/MCUs-Sales-To-Reach-RecordHigh-Annual-Revenues-Through-2022/ (accessed on 15 October 2022).
8. Cornetta, G.; Touhafi, A. Design and evaluation of a new machine learning framework for iot and embedded devices. *Electronics* **2021**, *10*, 600. [CrossRef]
9. Budjac, R.; Barton, M.; Schreiber, P.; Skovajsa, M. Analyzing Embedded AIoT Devices for Deep Learning Purposes. In Proceedings of the Computer Science On-Line Conference, 26 April 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 434–448.
10. Fedorov, I.; Adams, R.P.; Mattina, M.; Whatmough, P.N. SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers. *arXiv* **2019**, arXiv:1905.12107.
11. Zhang, Y.; Suda, N.; Lai, L.; Chandra, V. Hello Edge: Keyword Spotting on Microcontrollers. *arXiv* **2017**, arXiv:1711.07128.
12. Sakr, F.; Bellotti, F.; Berta, R.; De Gloria, A. Machine learning on mainstream microcontrollers. *Sensors* **2020**, *20*, 2638. [CrossRef]
13. Flamand, E.; Rossi, D.; Conti, F.; Loi, I.; Pullini, A.; Rotenberg, F.; Benini, L. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In Proceedings of the 2018 IEEE 29th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Milano, Italy, 10–12 July 2018; pp. 1–4. [CrossRef]
14. Warden, P. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv* **2018**, arXiv:1804.03209.
15. TinyML Foundation. Available online: https://www.tinyml.org/ (accessed on 15 October 2022).
16. Warden, P. Why the Future of Machine Learning Is Tiny, 2018b. Available online: https://petewarden.com/2018/06/11/why-the-future-of-machine-learning-is-tiny/ (accessed on 15 October 2022).
17. Lin, J.; Chen, W.M.; Lin, Y.; Gan, C.; Han, S. Mcunet: Tiny deep learning on iot devices. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 11711–11722.
18. Sanchez-Iborra, R.; Skarmeta, A.F. TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. *IEEE Circuits Syst. Mag.* **2020**, *20*, 4–18. [CrossRef]
19. Doyu, H.; Morabito, R.; Höller, J. Bringing machine learning to the deepest IoT edge with TinyML as-a-service. *IEEE IoT Newsl.* **2020**, *11*.
20. Kumar, A.; Goyal, S.; Varma, M. Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Precup, D., Teh, Y.W., Eds.; Volume 70, pp. 1935–1944.
21. Lai, L.; Suda, N.; Chandra, V. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs.*arXiv* **2018**, arXiv:1801.06601.
22. Garofalo, A.; Tagliavini, G.; Conti, F.; Rossi, D.; Benini, L. XpulpNN: Accelerating Quantized Neural Networks on RISC-V Processors through ISA Extensions. In Proceedings of the 23rd Conference on Design, Automation and Test in Europe, DATE'20,Grenoble, France, 9–13 March 2020; EDA Consortium: San Jose, CA, USA, 2020; pp. 186–191.
23. Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; Han, S. HAQ: Hardware-Aware Automated Quantization. *arXiv* **2018**, arXiv:1811.08886.
24. Moons, B.; Bankman, D.; Yang, L.; Murmann, B.; Verhelst, M. BinarEye: An Always-On Energy-Accuracy-Scalable Binary CNN Processor With All Memory On Chip in 28 nm CMOS. *arXiv* **2018**, arXiv:1804.05554.
25. Sudharsan, B.; Yadav, P.; Breslin, J.G.; Ali, M.I. An sram optimized approach for constant memory consumption and ultra-fast execution of ml classifiers on tinyml hardware. In Proceedings of the 2021 IEEE International Conference on Services Computing (SCC), Virtual, 5–11 September 2021; pp. 319–328.

26. MacGillivray, C.; Torchia, M. Internet of Things: Spending Trends and Outlook. 2019. Available online: https://www.idc.com/getdoc.jsp?containerId=US45161419 (accessed on 15 October 2022).
27. Gomez, J.; Patel, S.; Sarwar, S.S.; Li, Z.; Capoccia, R.; Wang, Z.; Pinkham, R.; Berkovich, A.; Tsai, T.H.; De Salvo, B.; et al. Distributed On-Sensor Compute System for AR/VR Devices: A Semi-Analytical Simulation Framework for Power Estimation. *arXiv* **2022**, arXiv:2203.07474.
28. Global Shipments of TinyML Devices to Reach 2.5 Billion by 2030. Available online: https://www.prnewswire.com/news-releases/global-shipments-of-tinyml-devices-to-reach-2-5-billion-by-2030--301123076.html (accessed on 15 October 2022).
29. Vuppalapati, C. *Democratization of Artificial Intelligence for the Future of Humanity*; CRC Press: Boca Raton, FL, USA, 2021.
30. Ray, P.P. A review on TinyML: State-of-the-art and prospects. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 1595–1623. [CrossRef]
31. Dutta, D.L.; Bharali, S. TinyML Meets IoT: A Comprehensive Survey. *Internet Things* **2021**, *16*, 100461. [CrossRef]
32. Wu, Z.; Jiang, M.; Li, H.; Zhang, X. Mapping the knowledge domain of smart city development to urban sustainability: A scientometric study. *J. Urban Technol.* **2021**, *28*, 29–53. [CrossRef]
33. Kim, H.; Chen, Q.; Yoo, T.; Kim, T.T.H.; Kim, B. A 1-16b Precision Reconfigurable Digital In-Memory Computing Macro Featuring Column-MAC Architecture and Bit-Serial Computation. In Proceedings of the ESSCIRC 2019-IEEE 45th European Solid State Circuits Conference (ESSCIRC), Cracow, Poland, 23–26 September 2019; pp. 345–348. [CrossRef]
34. Mahdavinejad, M.S.; Rezvan, M.; Barekatain, M.; Adibi, P.; Barnaghi, P.; Sheth, A.P. Machine learning for Internet of Things data analysis: A survey. *Digit. Commun. Netw.* **2018**, *4*, 161–175. [CrossRef]
35. Banbury, C.R.; Reddi, V.J.; Lam, M.; Fu, W.; Fazel, A.; Holleman, J.; Huang, X.; Hurtado, R.; Kanter, D.; Lokhmotov, A. Benchmarking TinyML Systems: Challenges and Direction. *arXiv* **2020**, arXiv:2003.04821.
36. Osman, A.; Abid, U.; Gemma, L.; Perotto, M.; Brunelli, D. TinyML Platforms Benchmarking. In Proceedings of the International Conference on Applications in Electronics Pervading Industry, Environment and Society, Genova, Italy, 26–27 September 2022; pp. 139–148.
37. Shafique, M.; Theocharides, T.; Reddy, V.J.; Murmann, B. TinyML: Current Progress, Research Challenges, and Future Roadmap. In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 5–9 December 2021; pp. 1303–1306.
38. Alongi, F.; Ghielmetti, N.; Pau, D.; Terraneo, F.; Fornaciari, W. Tiny Neural Networks for Environmental Predictions: An Integrated Approach with Miosix. In Proceedings of the 2020 IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Italy, 14–17 September 2020; pp. 350–355. [CrossRef]
39. Lootus, M.; Thakore, K.; Leroux, S.; Trooskens, G.; Sharma, A.; Ly, H. A VM/Containerized Approach for Scaling TinyML Applications. *arXiv* **2022**, arXiv:2202.05057.
40. Banbury, C.R.; Reddi, V.J.; Torelli, P.; Holleman, J.; Jeffries, N.; Király, C.; Montino, P.; Kanter, D.; Ahmed, S.; Pau, D.; et al. MLPerf Tiny Benchmark. *arXiv* **2021**, arXiv:2106.07597.
41. Ghamari, S.; Ozcan, K.; Dinh, T.; Melnikov, A.; Carvajal, J.; Ernst, J.; Chai, S. Quantization-Guided Training for Compact TinyML Models. *arXiv* **2021**, arXiv:2103.06231.
42. Coffen, B.; Mahmud, M. TinyDL: Edge Computing and Deep Learning Based Real-time Hand Gesture Recognition Using Wearable Sensor. In Proceedings of the 2020 IEEE International Conference on E-Health Networking, Application Services (HEALTHCOM), Virtual, 1–2 March 2021; pp. 1–6. [CrossRef]
43. Crocioni, G.; Gruosso, G.; Pau, D.; Denaro, D.; Zambrano, L.; di Giore, G. Characterization of Neural Networks Automatically Mapped on Automotive-grade Microcontrollers. *arXiv* **2021**, arXiv:2103.00201.
44. Disabato, S.; Roveri, M. Incremental On-Device Tiny Machine Learning. In Proceedings of the Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things, AIChallengeIoT'20, Virtual, 16–19 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 7–13. [CrossRef]
45. Doyu, H.; Morabito, R.; Brachmann, M. A TinyMLaaS Ecosystem for Machine Learning in IoT: Overview and Research Challenges. In Proceedings of the 2021 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 19–22 April 2021; pp. 1–5. [CrossRef]
46. Banbury, C.R.; Zhou, C.; Fedorov, I.; Navarro, R.M.; Thakker, U.; Gope, D.; Reddi, V.J.; Mattina, M.; Whatmough, P.N. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. *arXiv* **2020**, arXiv:2010.11267.
47. Fahim, F.; Hawks, B.; Herwig, C.; Hirschauer, J.; Jindariani, S.; Tran, N.; Carloni, L.P.; Guglielmo, G.D.; Harris, P.C.; Krupa, J.D.; et al. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. *arXiv* **2021**, arXiv:2103.05579.
48. Giordano, M.; Mayer, P.; Magno, M. 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, ENSsys'20, Virtual, 16–19 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 29–35. [CrossRef]
49. Kwon, J.; Park, D. Toward Data-Adaptable TinyML Using Model Partial Replacement for Resource Frugal Edge Device. In Proceedings of the The International Conference on High Performance Computing in Asia-Pacific Region, HPC Asia 2021, Virtual, 20–22 January 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 133–135. [CrossRef]
50. Langroudi, H.F.; Karia, V.; Pandit, T.; Kudithipudi, D. TENT: Efficient Quantization of Neural Networks on the tiny Edge with Tapered FixEd PoiNT. *arXiv* **2021**, arXiv:2104.02233.
51. Lin, J.; Chen, W.; Lin, Y.; Cohn, J.; Gan, C.; Han, S. MCUNet: Tiny Deep Learning on IoT Devices. *arXiv* **2020**, arXiv:2007.10319.

52. Roshan, A.N.; Gokulapriyan, B.; Siddarth, C.; Kokil, P. Adaptive Traffic Control With TinyML. In Proceedings of the 2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, India, 25–27 March 2021; pp. 451–455. [CrossRef]
53. Paul, A.J.; Mohan, P.; Sehgal, S. Rethinking Generalization in American Sign Language Prediction for Edge Devices with Extremely Low Memory Footprint. *arXiv* **2020**, arXiv:2011.13741.
54. Ren, H.; Anicic, D.; Runkler, T.A. The Synergy of Complex Event Processing and Tiny Machine Learning in Industrial IoT. *arXiv* **2021**, arXiv:2105.03371.
55. Sudharsan, B.; Salerno, S.; Nguyen, D.D.; Yahya, M.; Wahid, A.; Yadav, P.; Breslin, J.G.; Ali, M.I. TinyML Benchmark: Executing Fully Connected Neural Networks on Commodity Microcontrollers. In Proceedings of the 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), New Orleans, LA, USA, 14 June–31 July 2021; pp. 883–884. [CrossRef]
56. Svoboda, F.; Nunes, D.; Alizadeh, M.; Daries, R.; Luo, R.; Mathur, A.; Bhattacharya, S.; Silva, J.S.; Lane, N.D. Resource Efficient Deep Reinforcement Learning for Acutely Constrained TinyML Devices. Available online: https://openreview.net/forum?id=_vo8DFo9iuB (accessed on 15 October 2022).
57. Toma, C.; Popa, M.; Doinea, M. AI neural networks inference into the IoT embedded devices using tinyml for pattern detection within a security system. In Proceedings of the International Conference on Informatics in Economy Education, Research and Business Technologies, Timisoara, Romania, 21–24 May 2020; pp. 14–22.
58. Vuppalapati, C.; Ilapakurti, A.; Chillara, K.; Kedari, S.; Mamidi, V. Automating Tiny ML Intelligent Sensors DevOPS Using Microsoft Azure. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 10–13 December 2020; pp. 2375–2384. [CrossRef]
59. Vuppalapati, C.; Ilapakurti, A.; Kedari, S.; Vuppalapati, J.; Kedari, S.; Vuppalapati, R. Democratization of AI, Albeit Constrained IoT Devices amp; Tiny ML, for Creating a Sustainable Food Future. In Proceedings of the 2020 3rd International Conference on Information and Computer Technologies (ICICT), San Jose, CA, USA, 9–12 March 2020; pp. 525–530. [CrossRef]
60. Siddiqui, S.; Kyrkou, C.; Theocharides, T. Mini-NAS: A Neural Architecture Search Framework for Small Scale Image Classification Applications. 2021. Available online: https://openreview.net/forum?id=ERhIA5Y7IaT (accessed on 15 October 2022).
61. Jiao, B.; Zhang, J.; Xie, Y.; Wang, S.; Zhu, H.; Kang, X.; Dong, Z.; Zhang, L.; Chen, C. A 0.57-GOPS/DSP Object Detection PIM Accelerator on FPGA. In Proceedings of the 26th Asia and South Pacific Design Automation Conference, ASPDAC'21, Tokyo, Japan, 18–21 January 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 13–14. [CrossRef]
62. Tiny RespNet: A Scalable Multimodal TinyCNN Processor for Automatic Detection of Respiratory Symptoms. 2020. Available online: https://www.semanticscholar.org/paper/Tiny-RespNet%3A-A-Scalable-Multimodal-TinyCNN-for-of/2720facfbfed71d40a57dfb93b2845430b98cf67 (accessed on 15 October 2022).
63. Wen, X.; Famouri, M.; Hryniowski, A.; Wong, A. AttendSeg: A Tiny Attention Condenser Neural Network for Semantic Segmentation on the Edge. *arXiv* **2021**, arXiv:2104.14623.
64. Signoretti, G.; Silva, M.; Andrade, P.; Silva, I.; Sisinni, E.; Ferrari, P. An Evolving TinyML Compression Algorithm for IoT Environments Based on Data Eccentricity. *Sensors* **2021**, *21*, 4153. [CrossRef] [PubMed]
65. Capotondi, A.; Rusci, M.; Fariselli, M.; Benini, L. CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 871–875. [CrossRef]
66. David, R.; Duke, J.; Jain, A.; Reddi, V.J.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Regev, S.; et al. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. *arXiv* **2020**, arXiv:2010.08678.
67. de Prado, M.; Donze, R.; Capotondi, A.; Rusci, M.; Monnerat, S.; Benini, L.; Pazos, N. Robust navigation with TinyML for autonomous mini-vehicles. *arXiv* **2020**, arXiv:2007.00302.
68. Heim, L.; Biri, A.; Qu, Z.; Thiele, L. Measuring what Really Matters: Optimizing Neural Networks for TinyML. *arXiv* **2021**, arXiv:2104.10645.
69. Ren, H.; Anicic, D.; Runkler, T.A. TinyOL: TinyML with Online-Learning on Microcontrollers. *arXiv* **2021**, arXiv:2103.08295.
70. Wong, A.; Famouri, M.; Shafiee, M.J. AttendNets: Tiny Deep Image Recognition Neural Networks for the Edge via Visual Attention Condensers. *arXiv* **2020**, arXiv:2009.14385.
71. Zim, M.Z.H. TinyML: Analysis of Xtensa LX6 microprocessor for Neural Network Applications by ESP32 SoC. *arXiv* **2021**, arXiv:2106.10652.
72. Campolo, C.; Genovese, G.; Iera, A.; Molinaro, A. Virtualizing AI at the Distributed Edge towards Intelligent IoT Applications. *J. Sens. Actuator Netw.* **2021**, *10*, 13. [CrossRef]
73. Miao, H.; Lin, F.X. Enabling Large NNs on Tiny MCUs with Swapping. *arXiv* **2021**, arXiv:2101.08744.
74. Wong, A.; Famouri, M.; Pavlova, M.; Surana, S. TinySpeech: Attention Condensers for Deep Speech Recognition Neural Networks on Edge Devices. *arXiv* **2020**, arXiv:2008.04245.
75. Ajani, T.; Imoize, A.; Atayero, P.A. An Overview of Machine Learning within Embedded and Mobile Devices-Optimizations and Applications. *Sensors* **2021**, *21*, 4412. [CrossRef] [PubMed]
76. Mohan, P.; Paul, A.J.; Chirania, A. A Tiny CNN Architecture for Medical Face Mask Detection for Resource-Constrained Endpoints. *arXiv* **2020**, arXiv:2011.14858.
77. Warden, P.; Situnayake, D. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*; O'Reilly: Sebastopol, CA, USA, 2020.

78. Rusci, M.; Fariselli, M.; Capotondi, A.; Benini, L. Leveraging Automated Mixed-Low-Precision Quantization for tiny edge microcontrollers. *arXiv* **2020**, arXiv:2008.05124.
79. Soro, S. TinyML for Ubiquitous Edge AI. *arXiv* **2021**, arXiv:2102.01255.
80. Reddi, V.J.; Plancher, B.; Kennedy, S.; Moroney, L.; Warden, P.; Agarwal, A.; Banbury, C.R.; Banzi, M.; Bennett, M.; Brown, B.; et al. Widening Access to Applied Machine Learning with TinyML. *arXiv* **2021**, arXiv:2106.04008.
81. Han, H.; Siebert, J. TinyML: A Systematic Review and Synthesis of Existing Research. In Proceedings of the 2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Jeju Island, Republic of Korea, 21–24 February 2022; pp. 269–274. [CrossRef]
82. TensorFlow Lite (TFL). Available online: https://www.tensorflow.org/lite (accessed on 15 October 2022).
83. NanoEdge AI Studio. Available online: https://cartesiam-neai-docs.readthedocs-hosted.com/ (accessed on 15 October 2022).
84. PyTorch. Available online: https://pytorch.org/ (accessed on 15 October 2022).
85. uTensor. Available online: http://utensor.ai (accessed on 15 October 2022).
86. STM32Cube.AI. Available online: https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html (accessed on 15 October 2022).
87. Edge Impulse. Available online: https://www.edgeimpulse.com/ (accessed on 15 October 2022).
88. ELL. Available online: https://microsoft.github.io/ELL/ (accessed on 15 October 2022).
89. uTVM. Available online: https://octoml.ai/blog/tinyml-tvm-taming-the-final-ml-frontier (accessed on 15 October 2022).
90. uTVM System. Available online: https://tvm.apache.org/2020/06/04/tinyml-how-tvm-is-taming-tiny (accessed on 15 October 2022).
91. Wang, X.; Magno, M.; Cavigelli, L.; Benini, L. FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 4403–4417. [CrossRef]
92. Create, S.D.C.W. Petabytes of Data, What Are the Big Data Opportunities for the Car Industry. Available online: http://www.computerworlduk.com/news/data/boeing-787screate-half-terabyte-of-dataper-flight-says-virgin-atlantic-3433595/ (accessed on 7 December 2016).
93. Wang, S. Edge Computing: Applications, State-of-the-Art and Challenges. *Adv. Netw.* **2019**, *7*, 8–15. [CrossRef]
94. Mishra, P.; Puthal, D.; Tiwary, M.; Mohanty, S.P. Software defined IoT systems: Properties, state of the art, and future research. *IEEE Wirel. Commun.* **2019**, *26*, 64–71. [CrossRef]
95. Barani Sundaram, B.; Pandey, A.; Abiko, A.T.; Vijaykumar, J.; Rastogi, U.; Genale, A.H.; Karthika, P. Analysis of Machine Learning Data Security in the Internet of Things (IoT) Circumstance. In *Expert Clouds and Applications*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 227–236.
96. Puthal, D.; Mohanty, S.P.; Wilson, S.; Choppali, U. Collaborative edge computing for smart villages [energy and security]. *IEEE Consum. Electron. Mag.* **2021**, *10*, 68–71. [CrossRef]
97. Merenda, M.; Porcaro, C.; Iero, D. Edge machine learning for ai-enabled iot devices: A review. *Sensors* **2020**, *20*, 2533. [CrossRef] [PubMed]
98. Niu, W.; Ma, X.; Lin, S.; Wang, S.; Qian, X.; Lin, X.; Wang, Y.; Ren, B. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 16–20 March 2020; pp. 907–922.
99. Costa, R. The Internet of moving things [industry view]. *IEEE Technol. Soc. Mag.* **2018**, *37*, 13–14. [CrossRef]
100. Wollschlaeger, M.; Sauter, T.; Jasperneite, J. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Ind. Electron. Mag.* **2017**, *11*, 17–27. [CrossRef]
101. Fedorov, I.; Stamenovic, M.; Jensen, C.; Yang, L.C.; Mandell, A.; Gan, Y.; Mattina, M.; Whatmough, P.N. TinyLSTMs: Efficient Neural Speech Enhancement for Hearing Aids. In Proceedings of the Interspeech 2020, Shanghai, China, 25–29 October 2020. [CrossRef]
102. Rossi, D.; Conti, F.; Marongiu, A.; Pullini, A.; Loi, I.; Gautschi, M.; Tagliavini, G.; Capotondi, A.; Flatresse, P.; Benini, L. PULP: A parallel ultra low power platform for next generation IoT applications. In Proceedings of the 2015 IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, USA, 22–25 August 2015; pp. 1–39. [CrossRef]
103. Monfort Grau, M. TinyML: From Basic to Advanced Applications. Bachelor's Thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 2021.
104. Shanthamallu, U.S.; Spanias, A. Machine and Deep Learning Applications. In *Machine and Deep Learning Algorithms and Applications*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 59–72.
105. Bian, S.; Lukowicz, P. Capacitive sensing based on-board hand gesture recognition with TinyML. In Proceedings of the 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, Virtual, 21–26 September 2021; pp. 4–5.
106. Koufos, K.; El Haloui, K.; Dianati, M.; Higgins, M.; Elmirghani, J.; Imran, M.A.; Tafazolli, R. Trends in Intelligent Communication Systems: Review of Standards, Major Research Projects, and Identification of Research Gaps. *J. Sens. Actuator Netw.* **2021**, *10*, 60. [CrossRef]
107. Antonini, M.; Pincheira, M.; Vecchio, M.; Antonelli, F. A TinyML approach to non-repudiable anomaly detection in extreme industrial environments. In Proceedings of the 2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), Trento, Italy, 7–9 June 2022; pp. 397–402.

108. Venzke, M.; Klisch, D.; Kubik, P.; Ali, A.; Missier, J.D.; Turau, V. Artificial Neural Networks for Sensor Data Classification on Small Embedded Systems. *arXiv* **2020**, arXiv:2012.08403.
109. Pinge, A.; Bandyopadhyay, S.; Ghosh, S.; Sen, S. A Comparative Study between ECG-based and PPG-based Heart Rate Monitors for Stress Detection. In Proceedings of the 2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS), Bangalore, India, 4–8 January 2022; pp. 84–89. [CrossRef]
110. Tsoukas, V.; Boumpa, E.; Giannakas, G.; Kakarountas, A. A Review of Machine Learning and TinyML in Healthcare. In Proceedings of the 25th Pan-Hellenic Conference on Informatics, PCI 2021, Volos, Greece, 26–28 November 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 69–73. [CrossRef]
111. Boumpa, E.; Tsoukas, V.; Gkogkidis, A.; Spathoulas, G.; Kakarountas, A. Security and Privacy Concerns for Healthcare Wearable Devices and Emerging Alternative Approaches. In Proceedings of the International Conference on Wireless Mobile Communication and Healthcare, Virtual, 13–14 November 2021; Springer: Berlin/Heidelberg, Germany, 2022; pp. 19–38.
112. Zacharia, A.; Zacharia, D.; Karras, A.; Karras, C.; Giannoukou, I.; Giotopoulos, K.C.; Sioutas, S. An Intelligent Microprocessor Integrating TinyML in Smart Hotels for Rapid Accident Prevention. In Proceedings of the 2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), Ioannina, Greece, 23–25 September 2022; pp. 1–7. [CrossRef]
113. Santiago, P.R. Tinyml Monitoring Techniques for A-Vent: An Iot Edge for Tracking Clinical Risk Outcomes and Automatic Detection of Patient-Ventilator Asynchrony. Ph.D. Thesis, Ateneo de Manila University, Metro Manila, Philippines, 2021.
114. Nicolas, C.; Naila, B.; Amar, R.C. TinyML Smart Sensor for Energy Saving in Internet of Things Precision Agriculture platform. In Proceedings of the 2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN), Barcelona, Spain, 5–8 July 2022; pp. 256–259.
115. Ooko, S.O.; Ogore, M.M.; Nsenga, J.; Zennaro, M. TinyML in Africa: Opportunities and Challenges. In Proceedings of the 2021 IEEE Globecom Workshops (GC Wkshps), Madrid, Spain, 7–11 December 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
116. Ogore, M.M.; Nkurikiyeyezu, K.; Nsenga, J. Offline Prediction of Cholera in Rural Communal Tap Waters Using Edge AI inference. In Proceedings of the 2021 IEEE Globecom Workshops (GC Wkshps), Madrid, Spain, 7–11 December 2021; IEEE: Piscataway, NJ, USA, 2021, pp. 1–6.
117. PlantVillage. Available online: https://plantvillage.psu.edu/ (accessed on 15 October 2022).
118. Roasting Coffee To Perfection Using AI. Available online: https://highdemandskills.com/coffee-roasting-ai/ (accessed on 15 October 2022).
119. Santa, J.; Fernández, P.J. Seamless IPv6 connectivity for two-wheelers. *Pervasive Mob. Comput.* **2017**, *42*, 526–541. [CrossRef]
120. Andrade, P.; Silva, I.; Silva, M.; Flores, T.; Cassiano, J.; Costa, D.G. A TinyML Soft-Sensor Approach for Low-Cost Detection and Monitoring of Vehicular Emissions. *Sensors* **2022**, *22*, 3838. [CrossRef]
121. Tsoukas, V.; Gkogkidis, A.; Kampa, A.; Spathoulas, G.; Kakarountas, A. Enhancing Food Supply Chain Security through the Use of Blockchain and TinyML. *Information* **2022**, *13*, 213. [CrossRef]
122. Costa, D.G.; Peixoto, J.P.J.; Jesus, T.C.; Portugal, P.; Vasques, F.; Rangel, E.; Peixoto, M. A Survey of Emergencies Management Systems in Smart Cities. *IEEE Access* **2022**. [CrossRef]
123. Zhao, L.; Li, J.; Li, Q.; Li, F. A federated learning framework for detecting false data injection attacks in solar farms. *IEEE Trans. Power Electron.* **2021**, *37*, 2496–2501. [CrossRef]
124. Gkogkidis, A.; Tsoukas, V.; Papafotikas, S.; Boumpa, E.; Kakarountas, A. A TinyML-based system for gas leakage detection. In Proceedings of the 2022 11th International Conference on Modern Circuits and Systems Technologies (MOCAST), Samos, Greece, 3–7 July 2022; pp. 1–5. [CrossRef]
125. Alajlan, N.N.; Ibrahim, D.M. TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications. *Micromachines* **2022**, *13*, 851. [CrossRef]
126. Papageorgiou, E.I.; Theodosiou, T.; Margetis, G.; Dimitriou, N.; Charalampous, P.; Tzovaras, D.; Samakovlis, I. Short Survey of Artificial Intelligent Technologies for Defect Detection in Manufacturing. In Proceedings of the 2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA), Chania Crete, Greece, 12–14 July 2021; pp. 1–7. [CrossRef]
127. Delnevo, G.; Prandi, C.; Mirri, S.; Manzoni, P. Evaluating the practical limitations of TinyML: An experimental approach. In Proceedings of the 2021 IEEE Globecom Workshops (GC Wkshps), Madrid, Spain, 7–11 December; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
128. Miksikova, S.; Kuda, F.; Steinova, I. Intelligent and efficient parking solutions. *IOP Conf. Ser. Earth Environ. Sci.* **2021**, *900*, 012025. [CrossRef]
129. Zhou, A.; Muller, R.; Rabaey, J. Memory-Efficient, Limb Position-Aware Hand Gesture Recognition using Hyperdimensional Computing. *arXiv* **2021**, arXiv:2103.05267.
130. Mohan, P.; Paul, A.J.; Chirania, A. A tiny CNN architecture for medical face mask detection for resource-constrained endpoints. In *Innovations in Electrical and Electronic Engineering*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 657–670.
131. Häkkilä, J.; Lopes, P.; Kosch, T.; Nishida, J.; Strohmeier, P.; Abdelrahman, Y. Proceedings of the Augmented Humans Conference 2021: AHs 2021. Available online: https://research.ulapland.fi/en/publications/proceedings-augmented-humans-conference-2021-ahs-2021 (accessed on 15 October 2022).

132. Lord, M.; Kaplan, A. Mechanical Anomaly Detection on an Embedded Microcontroller. In Proceedings of the 2021 International Conference on Computational Science and Computational Intelligence (CSCI), Beijing, China, 4–6 December 2021; pp. 562–568. [CrossRef]
133. Nakhle, F.; Harfouche, A.L. Ready, Steady, Go AI: A practical tutorial on fundamentals of artificial intelligence and its applications in phenomics image analysis. *Patterns* **2021**, *2*, 100323. [CrossRef]
134. Curnick, D.J.; Davies, A.J.; Duncan, C.; Freeman, R.; Jacoby, D.M.; Shelley, H.T.; Rossi, C.; Wearn, O.R.; Williamson, M.J.; Pettorelli, N. SmallSats: A new technological frontier in ecology and conservation? *Remote Sens. Ecol. Conserv.* **2022**, *8*, 139–150. [CrossRef]
135. de Prado, M.; Rusci, M.; Capotondi, A.; Donze, R.; Benini, L.; Pazos, N. Robustifying the Deployment of TinyML Models for Autonomous Mini-Vehicles. *Sensors* **2021**, *21*, 1339. [CrossRef]
136. Vuletic, M.; Mujagic, V.; Milojevic, N.; Biswas, D. Edge AI Framework for Healthcare Applications. In Proceedings of the 30th International Joint Conference on Artificial Intelligence, Virtual, 19–26 August 2021.
137. Rana, A.; Dhiman, Y.; Anand, R. Cough Detection System using TinyML. In Proceedings of the 2022 International Conference on Computing, Communication and Power Technology (IC3P), Visakhapatnam, India, 7–8 January 2022; pp. 119–122.
138. Kwon, J.; Park, D. Hardware/Software Co-Design for TinyML Voice-Recognition Application on Resource Frugal Edge Devices. *Appl. Sci.* **2021**, *11*, 11073. [CrossRef]
139. Shamim, M.Z.M. Hardware Deployable Edge-AI Solution for Pre-screening of Oral Tongue Lesions using TinyML on Embedded Devices. *IEEE Embed. Syst. Lett.* **2022**, *14*, 183–186. [CrossRef]
140. TensorFlow Lite for Microcontrollers. Available online: https://www.tensorflow.org/lite/microcontrollers (accessed on 15 October 2022).
141. Intel. *Intel-64 and ia-32 Architectures Software Developer's Manual*; Volume 3A: System Programming Guide, Part 1 (64); Intel: Santa Clara, CA, USA, 2013.
142. Waterman, A.; Asanovi, K. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Document Version 20191213*; RISC-V Foundation; 2019.
143. Wu, X.; Lee, I.; Dong, Q.; Yang, K.; Kim, D.; Wang, J.; Peng, Y.; Zhang, Y.; Saligane, M.; Yasuda, M.; et al. A 0.04MM316NW Wireless and Batteryless Sensor System with Integrated Cortex-M0+ Processor and Optical Communication for Cellular Temperature Measurement. In Proceedings of the 2018 IEEE Symposium on VLSI Circuits, Honolulu, HI, USA, 18–22 June 2018; pp. 191–192. [CrossRef]
144. IC Insights Inc. *MCUs Expected to Make Modest Comeback After 2020 Drop*; IC Insights Inc.: Scottsdale, AZ, USA, 2020.
145. TensorFlow. TensorFlow Lite Guide, 2020b. Available online: https://www.tensorflow.org/lite/guide?hl=zh-cn (accessed on 15 October 2022).
146. Khandelwal, R. A Basic Introduction to TensorFlow Lite. Available online: https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292 (accessed on 15 October 2022).
147. Chettri, L.; Bera, R. A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems. *IEEE Internet Things J.* **2019**, *7*, 16–32. [CrossRef]
148. Liu, H.; Wei, Z.; Zhang, H.; Li, B.; Zhao, C. Tiny Machine Learning (Tiny-ML) for Efficient Channel Estimation and Signal Detection. *IEEE Trans. Veh. Technol.* **2022**, *71*, 6795–6800.
149. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet Things J.* **2017**, *4*, 1125–1142. [CrossRef]
150. Agiwal, M.; Roy, A.; Saxena, N. Next generation 5G wireless networks: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1617–1655. [CrossRef]
151. Bockelmann, C.; Pratas, N.; Nikopour, H.; Au, K.; Svensson, T.; Stefanovic, C.; Popovski, P.; Dekorsy, A. Massive machine-type communications in 5G: Physical and MAC-layer solutions. *IEEE Commun. Mag.* **2016**, *54*, 59–65. [CrossRef]
152. Palattella, M.R.; Dohler, M.; Grieco, A.; Rizzo, G.; Torsner, J.; Engel, T.; Ladid, L. Internet of things in the 5G era: Enablers, architecture, and business models. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 510–527. [CrossRef]
153. Kaloxylos, A.; Gavras, A.; Camps Mur, D.; Ghoraishi, M.; Hrasnica, H. AI and ML—Enablers for Beyond 5G Networks. 2020. Available online: https://www.recercat.cat/handle/2072/522533 (accessed on 15 October 2022).
154. Cayamcela, M.E.M.; Lim, W. Artificial intelligence in 5G technology: A survey. In Proceedings of the 2018 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 17–19 October 2018; pp. 860–865.
155. Karatzas, A.; Karras, A.; Karras, C.; Giotopoulos, K.C.; Oikonomou, K.; Sioutas, S. On Autonomous Drone Navigation Using Deep Learning and an Intelligent Rainbow DQN Agent. In Proceedings of the Intelligent Data Engineering and Automated Learning—IDEAL 2022, Manchester, UK, 24–26 November 2022; Yin, H., Camacho, D., Tino, P., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 134–145.
156. Zhang, C.; Patras, P.; Haddadi, H. Deep learning in mobile and wireless networking: A survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 2224–2287. [CrossRef]
157. Mao, Q.; Hu, F.; Hao, Q. Deep learning for intelligent wireless networks: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2595–2621. [CrossRef]
158. Luong, N.C.; Hoang, D.T.; Gong, S.; Niyato, D.; Wang, P.; Liang, Y.C.; Kim, D.I. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 3133–3174. [CrossRef]

159. Sanchez-Iborra, R. LPWAN and embedded machine learning as enablers for the next generation of wearable devices. *Sensors* **2021**, *21*, 5218. [CrossRef]
160. Karras, A.; Karras, C.; Giotopoulos, K.C.; Tsolis, D.; Oikonomou, K.; Sioutas, S. Peer to Peer Federated Learning: Towards Decentralized Machine Learning on Edge Devices. In Proceedings of the 2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), Ioannina, Greece, 23–25 September 2022; pp. 1–9. [CrossRef]
161. Singh Bali, M.; Gupta, K.; Kour Bali, K.; Singh, P.K. Towards energy efficient NB-IoT: A survey on evaluating its suitability for smart applications. *Mater. Today Proc.* **2022**, *49*, 3227–3234. [CrossRef]
162. Hossain, M.I.; Markendahl, J.I. Comparison of LPWAN Technologies: Cost Structure and Scalability. *Wirel. Pers. Commun.* **2021**, *121*, 887–903. [CrossRef]
163. Qin, J.; Li, Z.; Wang, R.; Li, L.; Yu, Z.; He, X.; Liu, Y. Industrial Internet of Learning (IIoL): IIoT based pervasive knowledge network for LPWAN—concept, framework and case studies. *CCF Trans. Pervasive Comput. Interact.* **2021**, *3*, 25–39. [CrossRef]
164. Khalifeh, A.; Aldahdouh, K.A.; Darabkh, K.A.; Al-Sit, W. A Survey of 5G Emerging Wireless Technologies Featuring LoRaWAN, Sigfox, NB-IoT and LTE-M. In Proceedings of the 2019 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET), Chennai, India, 21–23 March 2019; pp. 561–566. [CrossRef]
165. Ayoub, W.; Samhat, A.E.; Nouvel, F.; Mroue, M.; Prévotet, J.C. Internet of mobile things: Overview of lorawan, dash7, and nb-iot in lpwans standards and supported mobility. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 1561–1581.
166. Oliveira, L.; Rodrigues, J.J.; Kozlov, S.A.; Rabêlo, R.A.; Albuquerque, V.H.C.d. MAC layer protocols for Internet of Things: A survey. *Future Internet* **2019**, *11*, 16.
167. Shilpa, B.; Radha, R.; Movva, P. Comparative Analysis of Wireless Communication Technologies for IoT Applications. In *Proceedings of the Artificial Intelligence and Technologies*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 383–394.
168. Lim, W.Y.B.; Ng, J.S.; Xiong, Z.; Jin, J.; Zhang, Y.; Niyato, D.; Leung, C.; Miao, C. Decentralized edge intelligence: A dynamic resource allocation framework for hierarchical federated learning. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 536–550. [CrossRef]
169. Zhang, X.; Wang, Y.; Lu, S.; Liu, L.; Shi, W. OpenEI: An open framework for edge intelligence. In Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Richardson, TX, USA, 7–9 July 2019; pp. 1840–1851.
170. Zhang, Y.; Jiang, C.; Yue, B.; Wan, J.; Guizani, M. Information fusion for edge intelligence: A survey. *Inf. Fusion* **2022**, *81*, 171–186. [CrossRef]
171. Kamruzzaman, M.; Alrashdi, I.; Alqazzaz, A. New opportunities, challenges, and applications of edge-AI for connected healthcare in internet of medical things for smart cities. *J. Healthc. Eng.* **2022**, *2022*, 2950699. [CrossRef]
172. Ray, P.P.; Dash, D.; De, D. Edge computing for Internet of Things: A survey, e-healthcare case study and future direction. *J. Netw. Comput. Appl.* **2019**, *140*, 1–22. [CrossRef]