TinyML: The NextGen AI Technology for Standalone Devices

Satishkumar Kataria*[1] Pankaj Prajapati[2] Sachin Gajar[3] and Amit Rathod[4]

[1] Research Scholar, Gujarat Technological University, Chandkheda, Ahmedabad, Gujarat, India
[2] Associate Professor, Vishwakarma Government Engineering College, Gujarat, India
[3]Associate Professor, Nirma University, Ahmedabad, Gujarat, India
[4]Associate Professor, Government Engineering College, Bhavnagar, Gujarat, India
*Corresponding author: smkataria@gpahmedabad.ac.in

Abstract. Currently, the world is going through an AI and ML revolution, and we have seen tremendous growth in the implementation of AI in various sectors over the last decade. Conventional AI-based systems were implemented in a cloud-centric environment, using servers with high processing power, ample storage, and high-speed internet, which consume significant power. Tiny Machine Learning (TinyML) enables conventional ML models to run directly on resource-constrained embedded devices (i.e., Microcontrollers) with limited storage, processing capabilities, and power consumption. TinyML opens a new era to shift resource-hungry and cloud-centric conventional ML models to run in tiny and standalone resource-constrained devices. TinyML is a perfect solution for deploying AI and ML applications on sensors, wearables, IoT devices, and other small devices that are used in everyday life. In this paper, we have presented an intuitive review of the possibilities of TinyML. We have presented the background of TinyML, elaborated on TinyML-aware hardware platforms, tool sets for learning-to-deploy, the implementation methodology, and various use cases of TinyML. Finally, we have identified and discussed key challenges associated with this.

Keywords: TinyML, Edge AI, Microcontroller (MCU), Embedded System, Resource-constrained devices, Model Compression, Model Quantization.

1 Introduction

This section introduces Tiny Machine Learning (TinyML), a new paradigm that allows artificial intelligence (AI) to be directly implemented on embedded devices with limited resources, such as microcontrollers. It describes the fundamental ecology that underpins TinyML development and discusses the main application-driven and technological factors driving its uptake. These elements work together to provide the background and incentive for examining TinyML approaches, uses, and difficulties in the sections that follow.

S. Sharma et al. (eds.), *Proceedings of the International Conference on Recent Advances in Intelligent and Sustainable Technologies (RAIST 2026)*, Atlantis Highlights in Intelligent Systems 17,
https://doi.org/10.2991/978-94-6239-707-1_15

1.1 The TinyML Ecosystem

What is TinyML? TinyML is the intersection of ML and constrained devices[1], and now provides a more technological view of TinyML in the following Fig. 1. The diagram shown in Fig. 1 represents the TinyML ecosystem that enables deployment across devices, operating systems (OS), and AI-based software stacks. It shows the role of TinyML between different technologies and areas of intersection, which act as TinyML enablers. At each intersection point, TinyML will enable running an ML model on standalone, resource-constrained tiny devices (i.e., microcontrollers)[2].

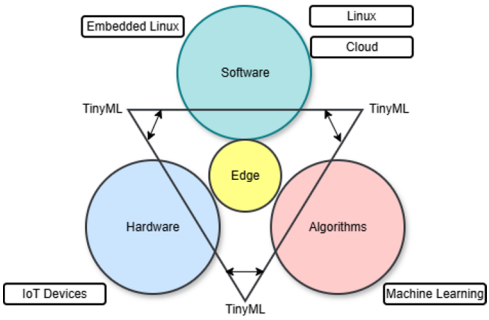

Fig. 1. TinyML enablers with different Technological areas. (Slightly modified version of Fig. 1. In [2]**)**

1.2 Drivers for TinyML adoption

We observed tremendous growth in Internet of Things (IoT) devices in recent years. It has been predicted that there will be 75 billion connected devices by 2025 as part of the Internet of Things (IoT) [3]. The majority of edge devices used with IoT-based systems are initially designed solely to collect sensor data and transmit it to a local or remote cloud [4]. This will generate a massive amount of data at the edge of networks. However, due to limitations in computing resources, connectivity, bandwidth, and power consumption, the biggest challenge is processing this data in real time on a cloud-based ML server and extracting meaningful insights [5]. IoT can help shift computation away from the cloud to network edge devices, leveraging collaboration among sensors, edge devices, and cloud facilities [6]. Furthermore, to avoid large amounts of data being transmitted and received by the IoT node, implement an ML model on the node itself. Conventional ML models mainly rely on cloud-based servers or high-end resource-rich computing platforms, which are not suitable for resource-constrained embedded systems [7]. Here, TinyML emerges as a solution to this problem. TinyML aims to deploy lightweight ML models directly on resource-constrained embedded devices without sacrificing efficiency, enabling devices to perform tasks locally without constant internet connectivity or external processing. The TinyML model is still in its early stages and requires proper alignment with existing edge-IoT frameworks. The original

research shows that the TinyML approach is crucial for the development of innovative IoT applications. In this paper, we discuss the background of the current state of TinyML, TinyML tool sets for model training-to-deployment, the elaboration of existing hardware platforms and software packages, methodologies for implementing TinyML, and use cases for TinyML across different applications. At last, we have identified key challenges for the TinyML research paradigm and discussed them.

2 Methodology for TinyML Implementation

This section provides a methodical overview of approaches for deploying TinyML solutions on hardware platforms with limited resources. The process of designing a system from start to finish is explained, followed by a discussion of popular software frameworks and development tools, and finally, a summary of the hardware platforms that enable TinyML deployment.

2.1 System Design Flow

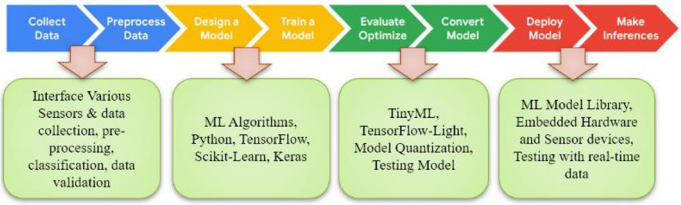

Fig. 2. TinyML Implementation Design Flow **[8]** (Slightly modified version then in **[8]**)

As shown in Fig. 2. System Design Flow for TinyML implementation is broadly divided into four steps ***(I) Data collection and preprocessing:*** This step divided in to Data collection phase and preprocessing phase in data collection phase raw signals, images, audio are captured from the real-world environment using different sensors with specific sampling rate, resolution, duration per sample, device placement, and environmental conditions [8] [9]. In the preprocessing phase, raw sensor streams are converted into model-ready inputs with a stable size and distribution, and it should be described as a repeatable pipeline. The common steps performed are: Cleaning and denoising, Segmentation, windowing, Resampling, Down sampling [9] [10], Normalization and scaling [10] [11]. A TinyML model can learn functional patterns and later run the same pipeline on-device. In TinyML, this step is crucial because sensor noise, limited memory, compute, and real-time constraints mean the dataset and the preprocessing pipeline must be designed together. ***(II) ML model design and training:*** In this step, we select (or create) an ML architecture that meets tiny-device constraints (Flash, RAM, latency), then define the training setup [8] [12]. ***(III) Testing model accuracy and quantization:*** evaluates the pre- and post-optimization performance of your TinyML model using both standard ML metrics and embedded-system metrics (latency, memory, energy), then applies quantization to shrink the model for microcontroller deployment while measuring the accuracy trade-off [8] [13]. Accuracy testing verifies

that the model generalizes to unseen data and meets deployment requirements by using a held-out test set that mimics real-world conditions (e.g., different users, noise levels, devices). In TinyML papers, accuracy testing goes beyond ML metrics to include on-device realism [14]. Model quantization: Reduces the precision of model weights and activations (e.g., from 32-bit floating-point to 8-bit integer) to cut model size, latency, and energy while fitting MCU limits, but it can degrade accuracy. Always report accuracy before/after quantization to show the trade-off [15]. ***(IV) Deploy model on an embedded device and conduct real-time testing:*** convert the trained/quantized model into microcontroller-compatible firmware, flash it to the target device, and validate performance under real-world conditions (latency, accuracy, power, stability). This final step ensures the model runs reliably in production, where factors like sensor noise, battery drain, and timing jitter can degrade performance [16]. Deployment means packaging the quantized model (.tflite) into firmware that runs inference in a continuous loop, triggered by timers, interrupts, or sensor events. The process uses embedded ML runtimes optimized for MCUs (no OS required) [17].

2.2 Software Tools for TinyML Development

TensorFlow Lite Micro (TFLite Micro): It is an open-source, lightweight, and efficient deep learning (DL) framework by Google for deploying ML models in resource-constrained edge devices such as microcontrollers (MCUs) and digital signal processors (DSPs), with only a few kilobytes of memory [18]. ***Edge Impulse Studio:*** Edge Impulse is a cloud-based end-to-end service platform for developing machine learning models in TinyML-based edge devices. It is a web-based cloud platform that provides one-step solutions for data collection, labelling, model training, model optimization, model deployment, integration, model testing, and management. Edge Impulse provides deployment options that let you export an inference package (often an SDK/library) and integrate it into embedded firmware or edge applications [19]. ***LiteRT for MCUs:*** It is a C++-based runtime library used for deploying and executing ML models on MCU-class hardware (TinyML-based devices). It is specifically written and tested for 32-bit ARM Cortex-M processors and has also been ported to architectures like the ESP32. The supported development platforms and boards are Arduino Nano 33 BLE Sense, SparkFun Edge, Espressif ESP32, ESP-EYE, STM32F Discovery, Adafruit EdgeBadge, a set of supported development boards and platforms, including Arduino Nano 33 BLE Sense, SparkFun Edge, STM32F746 Discovery kit, Adafruit EdgeBadge, Espressif ESP32-DevKitC, ESP-EYE, Wio Terminal (ATSAMD51), Sony Spresense, and Himax WE I Plus EVB [20]. ***STM32Cube.AI:*** It is developed for STM32F microcontrollers from ST Microelectronics. The STMicroelectronics toolchain will handle integrating the trained model's STM32-optimized library into the firmware project. This is designed to compile, optimize, and evaluate Edge AI models for STM32F microcontrollers, and to support ST Microelectronics' Natural ART NPU accelerator [21]. ***CMSIS-NN:*** It is developed by ARM for ARM Cortex-M microcontrollers as a library of highly optimized neural network building blocks. So that the common Neural network layers (Quantized with int8) run efficiently and faster than generic C implementations [22]. ***µTVM:*** It is a framework for ML model compilation and extends TVM to

support Microcontroller devices by adding runtime and backend components. µTVM is used for host-driven execution. The Mainframe PC hosts the controls and compiles, builds, and flashes firmware to the device over a communication link. It supports model compilation, automated performance optimization, and real-world microcontroller measurements. To measure timing results, µTVM can be paired with Auto-TVM to improve real-world latency on MCU targets [23], [24]. ***NVIDIA TensorRT:*** This is an SDK used for high-performance inference on NVIDIA GPUs. It is used to train and optimize a model to run faster and more efficiently during deployment. TensorRT mainly focuses on inference optimization by restructuring the model graph and selecting fast GPU kernels [25]. ***ONNX Runtime Mobile:*** ONNX Runtime Mobile enables on-device inference of ML models in a limited-size mobile application. Model inference will be generated by ONXX Runtime Mobile for Android (ONXXRUNTIME – Android), iOS (ONXXRUNTIME -C), and for .NET options (MAUI, Xamarin) on devices [26].

2.3 Hardware Platforms for TinyML Development

There are many platforms available from different vendors. Here are some compelling platforms, reviewed with their specifications, connectivity options, and available sensors. Arduino Nano 33 BLE Sense [27], Apollo3 [28], STM32F Discovery [29], ESP32 [30], ESP-EYE [31], Sony's Spresense [32], GAP8 [33], OpenMV Cam H7 Plus [34], Nordic Semi nRF52840 DK [35], CC1352P Launchpad [36], Pico4ML BLE [37], Raspberry Pi 5 [38].

3 Applications of TinyML

This section explores the practical applicability of TinyML across diverse domains, highlighting why local on-device inference is preferred over cloud-based processing in many scenarios. It further reviews representative TinyML application ecosystems by analyzing prior research works, application categories, and reported performance outcomes.

3.1 Run TinyML Locally

Why run TinyML locally? To justify the answer, there are three main reasons: latency (in response time), data privacy, and device power consumption. ***Reducing latency:*** Sending data to the cloud and receiving it back are not instant, which affects the application's response time. So, when we use the cloud, latency exists; in fact, real-time applications must respond reliably. ***Reducing power consumption:*** Sending all sensor data to the cloud/server is not a power-efficient approach, especially in battery-operated, power-constrained devices [39], [40].

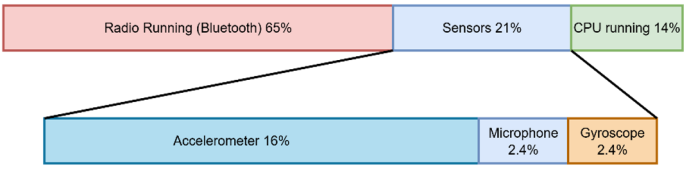

Fig. 3. Power consumption breakdown for the Arduino Nano 33 BLE Sense board [39]

Fig. 3 shows the percentage power consumption breakdown of the Arduino Nano 33 BLE Sense board's onboard components. We can see that 65% of the power is used by Bluetooth communication, 21% by the onboard, and 14% by the CPU. As we can see, CPU computation is more power-efficient than Bluetooth communication (14% vs 65%), so it is preferable to compute more and transmit less. That is why the importance of running TinyML locally ignites their need for the most energy-efficient typical embedded systems. ***Privacy:*** Running a local ML model means preserving the user's privacy and avoiding sharing sensitive information[39], [40].

3.2 TinyML Ecosystem

Fig. 4. TinyML Ecosystem [41]

TinyML-enabled devices are primarily designed to operate as standalone devices without cloud or internet connectivity [42]. This approach opens scope of research in many areas to make portable standalone devices like (1) Smart Healthcare (2) Industry 4.0 (3) Environment Monitoring (4) Smart Home (5) Vehicular (6) Smart city. [41]. Adoption of IoT devices growing rapidly and to make IoT Infrastructure power efficient, secure and faster the TinyML will play a vital role as a backbone of this system. TinyML makes end devices more intelligent and capable of decision-making, leading to significant savings in resources (e.g., processing power, storage, communication power). Several applications already exist and are presented in various publications, and many more research areas are being conducted.

Table 1. Review of applications implemented on TinyML

Ref	Application	Methodology used	Output
[43]	Real-Time Fall Detection	RNN Architectures RNN architecture was made in TensorFlow of microcontroller units (MCU), Tri-axial accelerometers as sensors, SisFall dataset, 80%/20% train/test split	Achieved 98% accuracy, prototype present.
[44]	Cough Detection	Used Arduino 33 BLE Sense board, prepared Audio Mel Frequency Cepstral Coefficients of all the audio windows, trained neural network using Keras library in edge impulse tool.	Achieved 96.9% accuracy for TinyML based modeling.
[45]	Prediction of Chronic Obstructive Pulmonary Disease using Synthetic Exhaled Breath Data-Based	Mostly.ai platform used to generate synthetic data set generation, COPD open data set used, Sensor data collected from SP3, MQ3, TGS822, MQ133, TGS813, TGS800, MG135, Neural Network created using Keras, K-mean algorithm for features extraction. Target Board used STM32	With COPD dataset two model trained one got accuracy 96.9% and other 92.4%
[46]	Adaptive Traffic Control	detects vehicles with piezoelectric sensors, The two-point time ratio method is utilized to identify the vehicles using the sensor's data, reset traffic signal timers according to road traffic adaptively using combination of SOTA TinyML, Random Forest Prediction algorithm.	Achieved Traffic aware signaling on road. No accuracy stated.
[47]	Heart rate estimation	Used Photoplethysmography (PPG) signals, Gaussian-based weighting function with the dual-level mechanism (GAUL), convolutional neural networks (CNN), dataset PPG-DaLiA	They evaluate on three benchmark datasets (PPG-DaLiA, IEEE SPC Train, IEEE SPC Test), GAUL achieves the best results.
[48]	Stress Detection	Processor Nordic RF52832, RISC-V based SoC, 9-axis motion sensor, Pressure sensor, microphone, ECG, EMG & bioimpedance AFE (Maxim MAX30001), galvanic skin response (GSR) front end, TinyML model designed using multi-layer artificial neural networks	24 stress classifications per minute in indoor conditions.
[49]	Gas leakage detection	Arduino 33 Sense BLE board, MQ series gas sensors, LCD, used Edge Impulse Studio to train the model, Anomaly Detection learning block based on K-Means clustering.	two test cases, smoke F1 = 0.77 and Ammonia F1 = 0.70.

3.3 Computational Efficiency of TinyML Models

Due to stringent limitations on processor power, memory, and energy, resource-constrained devices lead to computational complexity and inference delay, which are the most critical performance parameters in TinyML systems. In microcontroller-based systems, computational complexity is commonly quantified in terms of model parameters and multiply accumulate (MAC) operations, both of which directly affect execution time and power consumption. TinyML models use reduced-precision arithmetic, lightweight architectures, and model optimization strategies such as quantization and pruning to reduce computational cost. Since most TinyML models only require a few thousand to a few million MACs per inference, they can be implemented on low-power MCUs. Several variables, including clock frequency, memory access efficiency, and instruction-level optimizations, affect inference time. Real-time or near-real-time operation is supported by optimized TinyML models, which on representative systems such as ESP32-based microcontrollers and ARM Cortex-M often achieve inference latencies ranging from a few milliseconds to many tens of milliseconds. The TinyML-based method offers lower computational complexity and bounded inference latency compared to traditional deep learning techniques, demonstrating its applicability and practical efficiency for continuous on-device inference in resource-constrained environments. Table 2 presents a comparative analysis of the effects on various parameters across different deployment platforms [8], [50], [51], [52], [53].

Table 2. Numerical Comparison of Cloud, Edge AI, and TinyML Approaches [8],[50],[51],[52],[53]

Metric	Cloud AI	Edge AI (SoC)	TinyML (MCU)
Model Size	>10 MB	1-10 MB	<500 KB
MACs / Inference	>100 M	10-50 M	<5 M
Inference Latency	>100 ms	20-50 ms	2-5 ms
Energy / Inference	High	Medium	Low
Hardware	GPU / TPU	NPU-enabled SoC	MCU only
Always-on-capability	No	Limited	Yes

4 Key Challenges Associated with TinyML

Development with TinyML is challenging mainly because devices run ML inference on MCUs with tight constraints on computation power, energy, memory, and real-time responsiveness. These limitations force trade-offs among parameters such as accuracy, latency, power, and deployability; this is not the case with cloud-centric or desktop ML models [54]. The key challenges are explained below: ***Resource-constrained devices:*** MCUs having limited memory resources in terms of hundreds of KBs or a few MBs.

So it is tough to manage model weights, intermediate activations, and runtime buffer allocation. In TinyML, achieving the same performance after aggressive memory optimization remains a significant challenge [55]. ***Reduced Throughput:*** MCUs lack MACs, GPUs, or NPUs, so they cannot handle heavy computational workloads. Therefore, lightweight models with fewer layers and smaller input sizes need to be designed, which is the opposite of the cloud-centric approach [12]. ***Energy constrained:*** MCUs are operated at milliwatt power in this scenario to manage power and performance. A new approach, such as duty cycling, less pre- and post-processing, and model quantization techniques, must be used to work with TinyML [56]. ***Challenges in Model optimization:*** When we shrink the model through quantization to improve speed and energy efficiency, accuracy is reduced. Hence, choosing trade-offs and their limits is a critical consideration [57], [58]. ***TinyML model deployment and Runtime issues:*** There are so many platforms and software tools to work with TinyML in this scenario converting model to light weight form it passes through many processes like training framework in separate tool after that model compression and quantization process then C, C++ library integration with source code and build embedded code all this needs various cross platforms which difficulties in debugging process. Some Real-time applications require predictable response times due to jitter in interrupts, which may manipulate buffer memory and sensor I/O, resulting in missed response deadlines [59]. ***Data Miss-matching:*** Datasets prepared with the help of various sensors in specific environmental conditions. When environmental conditions change, sensor readings may vary, affecting model accuracy. In some cases, a firmware upgrade may be needed, which is a significant engineering challenge [60]. ***Data security and privacy:*** Though on-device execution of the model reduces data transmission and energy consumption, there is a risk of model tampering or extraction. In addition to the above, there are many other challenges, including hardware and software heterogeneity, a lack of benchmarking tools, a lack of datasets, a lack of a widely accepted method, scattered edge computing infrastructure, edge platform orchestration, data and network management, and reliability issues [61].

5 Conclusion and Future Scope

In the last few years, the field of AI has grown, creating a need for TinyML, which has also increased its importance for implementing on-device intelligence in ultra-low-power devices. TinyML enables the creation of locally intelligent devices by running inference on the whole model without cloud connectivity, thereby addressing the key challenges of conventional cloud-centric models (latency, data privacy, bandwidth limitations, and power consumption). In this paper, we have conducted an intensive review of various papers/materials and have presented a systematic review of TinyML technology, covering the needs of TinyML, a step-by-step implementation methodology, comparisons of various software tools and hardware platforms, and TinyML ecosystems and their application areas. In the last phase of the paper, we addressed key challenges associated with this, including limited resource budgets, the lack of standardized benchmarking methodologies, and the trade-off between model accuracy and memory

size. difficulties in updating deployed models; sensor data is affected by environmental parameters, which leads to reduced model accuracy and performance, etc. These many challenges present a future research scope, especially to develop on-device learning technology, benchmark the technology, enable over-the-air firmware upgrades and deployments, and improve model quantization vs. accuracy trade-offs.

References

1. T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications," *Sensors*, vol. 21, no. 13, p. 4412, Jan. 2021, doi: 10.3390/s21134412.
2. H. Doyu, R. Morabito, and M. Brachmann, "A TinyMLaaS Ecosystem for Machine Learning in IoT: Overview and Research Challenges," in *2021 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, Hsinchu, Taiwan: IEEE, Apr. 2021, pp. 1–5. doi: 10.1109/VLSI-DAT52063.2021.9427352.
3. B. Sudharsan *et al.*, "OTA-TinyML: Over the Air Deployment of TinyML Models and Execution on IoT Devices," *IEEE Internet Comput.*, vol. 26, no. 3, pp. 69–78, May 2022, doi: 10.1109/MIC.2021.3133552.
4. J. Liu, C. Liu, B. Wang, G. Gao, and S. Wang, "Optimized Task Allocation for IoT Application in Mobile-Edge Computing," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10370–10381, Jul. 2022, doi: 10.1109/JIOT.2021.3091599.
5. T. Bouguera, J.-F. Diouris, J.-J. Chaillout, R. Jaouadi, and G. Andrieux, "Energy Consumption Model for Sensor Nodes Based on LoRa and LoRaWAN," *Sensors*, vol. 18, no. 7, p. 2104, Jun. 2018, doi: 10.3390/s18072104.
6. J. Singh, Y. Bello, A. R. Hussein, A. Erbad, and A. Mohamed, "Hierarchical Security Paradigm for IoT Multiaccess Edge Computing," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5794–5805, Apr. 2021, doi: 10.1109/JIOT.2020.3033265.
7. C. Nicolas, B. Naila, and R.-C. Amar, "TinyML Smart Sensor for Energy Saving in Internet of Things Precision Agriculture platform," in *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, Barcelona, Spain: IEEE, Jul. 2022, pp. 256–259. doi: 10.1109/ICUFN55119.2022.9829675.
8. V. J. Reddi *et al.*, "Widening Access to Applied Machine Learning with TinyML," Jun. 09, 2021, *arXiv*: arXiv:2106.04008. doi: 10.48550/arXiv.2106.04008.
9. R. Samanta, B. Saha, S. K. Ghosh, and R. B. Roy, "Optimizing TinyML: The Impact of Reduced Data Acquisition Rates for Time Series Classification on Microcontrollers," Sep. 17, 2024, *arXiv*: arXiv:2409.10942. doi: 10.48550/arXiv.2409.10942.
10. "TensorFlow Blog." [Online]. Available: https://blog.tensorflow.org/2021/05/building-tinyml-application-with-tf-micro-and-sensiml.html
11. T. Langer, M. Widra, and V. Beyer, "TinyML Towards Industry 4.0: Resource-Efficient Process Monitoring of a Milling Machine," Aug. 22, 2025, *arXiv*: arXiv:2508.16553. doi: 10.48550/arXiv.2508.16553.

12. S. Heydari and Q. H. Mahmoud, "Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions," *Sensors*, vol. 25, no. 10, p. 3191, May 2025, doi: 10.3390/s25103191.
13. A. Dey, S. Srivastava, G. Singh, and R. G. Pettit, "Real-Time Performance Benchmarking of TinyML Models in Embedded Systems (PICO: Performance of Inference, CPU, and Operations)," Sep. 05, 2025, *arXiv*: arXiv:2509.04721. doi: 10.48550/arXiv.2509.04721.
14. G. Delnevo, S. Mirri, C. Prandi, and P. Manzoni, "An evaluation methodology to determine the actual limitations of a TinyML-based solution," *Internet of Things*, vol. 22, p. 100729, Jul. 2023, doi: 10.1016/j.iot.2023.100729.
15. "LearnOpenCV." [Online]. Available: https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/
16. "hackster.io." [Online]. Available: https://www.hackster.io/news/easy-tinyml-on-esp32-and-arduino-a9dbc509f26c
17. "ELOQUENT ARDUINO." [Online]. Available: https://eloquentarduino.com/posts/tensorflow-lite-tinyml-esp32
18. "GITHUB." [Online]. Available: https://github.com/tensorflow/tflite-micro
19. "EDGE IMPULSE." [Online]. Available: https://docs.edgeimpulse.com/studio/projects/deployment
20. "Google AI for Developers." [Online]. Available: https://ai.google.dev/edge/litert/microcontrollers/overview
21. "ST Microelectronics." [Online]. Available: https://stm32ai.st.com/stm32-cube-ai/
22. G. Armeniakos, G. Mentzos, and D. Soudris, "Accelerating TinyML Inference on Microcontrollers through Approximate Kernels," Sep. 25, 2024, *arXiv*: arXiv:2409.16815. doi: 10.48550/arXiv.2409.16815.
23. "Apache TVM." [Online]. Available: https://tvm.apache.org/2020/06/04/tinyml-how-tvm-is-taming-tiny
24. "daobook.github.io." [Online]. Available: https://daobook.github.io/tvm/docs/arch/microtvm_design.html
25. "NVIDIA TensorRT Documentation." [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/latest/index.html
26. "ONNX RUNTIME." [Online]. Available: https://onnxruntime.ai/docs/tutorials/mobile/
27. "Arduino official website." [Online]. Available: https://docs.arduino.cc/hardware/nano-33-ble-sense/
28. "Sparkfun Electronics." [Online]. Available: https://www.sparkfun.com/sparkfun-edge-development-board-apollo3-blue.html
29. "STMicroelectronics." [Online]. Available: https://www.st.com/en/evaluation-tools/stm32f4discovery.html
30. "ESPRESSIF." [Online]. Available: https://docs.espressif.com/projects/esp-devkits/en/latest/esp32/esp32-devkitc/index.html
31. "ESPRESSIF." [Online]. Available: https://www.espressif.com/en/products/devkits/esp-eye/overview
32. "SONY." [Online]. Available: https://developer.spresense.sony-semicon.com/

33. E. Flamand *et al.*, "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Milan: IEEE, Jul. 2018, pp. 1–4. doi: 10.1109/ASAP.2018.8445101.
34. "OpenMV." [Online]. Available: https://openmv.io/products/openmv-cam-h7-plus?srsltid=AfmBOorEwS-INqVXm8RV3ze7xfHHyw3ipE43skpIoAW6Mmes06tMdL5pH
35. "NORDIC Semiconductor." [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/nRF52840-DK
36. "TEXAL INSTRUMENT." [Online]. Available: https://www.ti.com/tool/LAUNCHXL-CC1352P
37. "Raspberry Pi." [Online]. Available: https://forums.raspberrypi.com/viewtopic.php?t=324642&sid=dacf1774122e6bebd9a1d523722e2068
38. "Raspberry Pi." [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/
39. G. M. Iodice, *TinyML Cookbook: Combine artificial intelligence and ultra-low-power embedded devices to make the world smarter*, 1st ed. Birmingham: Packt Publishing Limited, 2022.
40. B. Sudharsan *et al.*, "TinyML Benchmark: Executing Fully Connected Neural Networks on Commodity Microcontrollers," in *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, New Orleans, LA, USA: IEEE, Jun. 2021, pp. 883–884. doi: 10.1109/WF-IoT51360.2021.9595024.
41. R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities," *IEEE Circuits Syst. Mag.*, vol. 20, no. 3, pp. 4–18, 2020, doi: 10.1109/MCAS.2020.3005467.
42. C. R. Banbury *et al.*, "Benchmarking TinyML Systems: Challenges and Direction," 2020, *arXiv*. doi: 10.48550/ARXIV.2003.04821.
43. E. Torti *et al.*, "Embedded Real-Time Fall Detection with Deep Learning on Wearable Devices," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, Prague: IEEE, Aug. 2018, pp. 405–412. doi: 10.1109/DSD.2018.00075.
44. A. Rana, Y. Dhiman, and R. Anand, "Cough Detection System using TinyML," in *2022 International Conference on Computing, Communication and Power Technology (IC3P)*, Visakhapatnam, India: IEEE, Jan. 2022, pp. 119–122. doi: 10.1109/IC3P52835.2022.00032.
45. S. O. Ooko, D. Mukanyiligira, J. P. Munyampundu, and J. Nsenga, "Synthetic Exhaled Breath Data-Based Edge AI Model for the Prediction of Chronic Obstructive Pulmonary Disease," in *2021 International Conference on Computing and Communications Applications and Technologies (I3CAT)*, Ipswich, United Kingdom: IEEE, Sep. 2021, pp. 1–6. doi: 10.1109/I3CAT53310.2021.9629420.
46. A. N. Roshan, B. Gokulapriyan, C. Siddarth, and P. Kokil, "Adaptive Traffic Control With TinyML," in *2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Chennai, India: IEEE, Mar. 2021, pp. 451–455. doi: 10.1109/WiSPNET51692.2021.9419472.

47. J. Kim, H. Cho, M. Lee, and S. B. Kim, "Multi-source partial domain adaptation with Gaussian-based dual-level weighting for PPG-based heart rate estimation," *Knowledge-Based Systems*, vol. 309, p. 112769, Jan. 2025, doi: 10.1016/j.knosys.2024.112769.
48. M. Magno, X. Wang, M. Eggimann, L. Cavigelli, and L. Benini, "InfiniWolf: Energy Efficient Smart Bracelet for Edge Computing with Dual Source Energy Harvesting," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2020, pp. 342–345. doi: 10.23919/DATE48585.2020.9116218.
49. A. Gkogkidis, V. Tsoukas, S. Papafotikas, E. Boumpa, and A. Kakarountas, "A TinyML-based system for gas leakage detection," in *2022 11th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, Bremen, Germany: IEEE, Jun. 2022, pp. 1–5. doi: 10.1109/MOCAST54814.2022.9837510.
50. M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017, doi: 10.1109/MC.2017.9.
51. P. Warden and D. Situnayake, *TinyML: machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*, First edition. Bejing Boston Farnham Sebastopol Tokyo: O'Reilly, 2020.
52. Y. Kang *et al.*, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, Xi'an China: ACM, Apr. 2017, pp. 615–629. doi: 10.1145/3037697.3037698.
53. N. N. Alajlan and D. M. Ibrahim, "TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications," *Micromachines*, vol. 13, no. 6, p. 851, May 2022, doi: 10.3390/mi13060851.
54. G. Delnevo, C. Prandi, S. Mirri, and P. Manzoni, "Evaluating the practical limitations of TinyML: an experimental approach," in *2021 IEEE Globecom Workshops (GC Wkshps)*, Madrid, Spain: IEEE, Dec. 2021, pp. 1–6. doi: 10.1109/GCWkshps52748.2021.9682101.
55. L. Capogrosso, F. Cunico, D. S. Cheng, F. Fummi, and M. Cristani, "A Machine Learning-oriented Survey on Tiny Machine Learning," Sep. 26, 2023, *arXiv*: arXiv:2309.11932. doi: 10.48550/arXiv.2309.11932.
56. N. Schizas, A. Karras, C. Karras, and S. Sioutas, "TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review," *Future Internet*, vol. 14, no. 12, p. 363, Dec. 2022, doi: 10.3390/fi14120363.
57. R. Immonen and T. Hämäläinen, "Tiny Machine Learning for Resource-Constrained Microcontrollers," *Journal of Sensors*, vol. 2022, pp. 1–11, Nov. 2022, doi: 10.1155/2022/7437023.
58. C. M. Bhushan *et al.*, "Deploying TinyML for energy-efficient object detection and communication in low-power edge AI systems," *Sci Rep*, vol. 15, no. 1, p. 44299, Dec. 2025, doi: 10.1038/s41598-025-27818-9.
59. Y. Shi, E. Njor, P. Martínez-Nuevo, S. E. Shepstone, and X. Fafoutis, "Data Aware Differentiable Neural Architecture Search for Tiny Keyword Spotting

Applications," Jul. 21, 2025, *arXiv*: arXiv:2507.15545. doi: 10.48550/arXiv.2507.15545.

60. J. D. Velasquez, L. Cadavid, and C. J. Franco, "Emerging trends and strategic opportunities in tiny machine learning: A comprehensive thematic analysis," *Neurocomputing*, vol. 648, p. 130746, Oct. 2025, doi: 10.1016/j.neucom.2025.130746.
61. J. Huckelberry, Y. Zhang, A. Sansone, J. Mickens, P. A. Beerel, and V. J. Reddi, "TinyML Security: Exploring Vulnerabilities in Resource-Constrained Machine Learning Systems," Nov. 11, 2024, *arXiv*: arXiv:2411.07114. doi: 10.48550/arXiv.2411.07114.