Frontiers of Multidisciplinary Exploration
Volume 2, Issue 2, September 2026, Page 1-7
ISSN: 3139-4086

TinyML for IoT Sensing: A Comparative Review of Frameworks for Deploying Machine-Learning Models on Low-Power Microcontrollers

Sidharth Kashyap[1]

[1]Assistant Professor, Department of Mechanical Engineering, Adi Sankara University, Andhra Pradesh, India

Corresponding Author Email: Kashyap.pce99@gmail.com

Received 6 April 2026, accepted 30 June 2026, Published online: 30 September 2026

Abstract

The Internet of Things (IoT) generates vast amounts of data from sensors, but sending raw data to the cloud for analysis adds latency, bandwidth and energy costs and privacy considerations. To overcome these limitations, Tiny machine learning (TinyML) is being developed that runs inference directly on the microcontroller unit (MCU), which has a few hundred kilobytes of memory and a few milliwatts power budget. This paper reviews and compares some of the main solutions that have been developed for deploying machine-learning (ML) models on low-power microcontrollers (MCUs): TensorFlow Lite for Microcontrollers (LiteRT), CMSIS-NN, and STM32Cube.Together with model compression methods for which they rely, such as integer quantization, pruning and neural architecture search, AI (ST Edge AI) and Edge Impulse and the MCUNet/TinyEngine co-design approach. The frameworks are contrasted with regard to memory consumption, inference latency, energy consumption per inference, accuracy loss upon compression, and hardware portability and usability. The examined evidence reveals that vendor-optimised kernels and compiler-based solutions perform best when it comes to latency and memory consumption on the target machine while interpreter-based and platform-based solutions are more portable and provide quick development. Significant challenges are (1) the lack of consistency in how benchmarks are performed, (2) the lack of support for learning on-device, (3) the hardware and toolchain fragmentation and (4) maintaining accuracy in extreme memory constraints. The review identifies hardware–software co-design and the use of a standardised benchmark like MLPerf Tiny as key pillars towards scaling TinyML from prototypes to reliable, large-scale IoT deployments.

Keywords: tiny machine learning (TinyML); microcontrollers; edge AI; Internet of Things; model compression; quantisation; embedded inference

1. Introduction

The Internet of Things (IoT) is the connection across billions of sensing devices in industrial, agricultural, healthcare and smart city applications. Most of these devices are based on low-cost processors like microcontroller units (MCUs) like the Arm Cortex-M family, Espressif ESP32, and new RISC-V devices that operate for months or years on batteries or harvested energy [1]. In the traditional IoT setup, these entities can be seen as passive data loggers that send raw data measurements to a gateway or cloud-based server for processing. The cloud-based approach has its drawbacks. A major energy expense on a sensor node is often wireless

transmission, and streaming raw audio, vibration or image data quickly wastes the battery life of the node. The added round-trip delay is prohibitive for time-sensitive applications, like machine fault detection, and transmitting personal data off the device presents privacy and security concerns. Some remote installations might also have intermittent or no connectivity. A different approach is Tiny machine learning (TinyML): transfer only results or events since machine-learning (ML) inference is executed on the MCU itself [2]. The biggest challenge is that of the resource limitations of the target hardware, which are typically in the order of 1 MB or less of flash memory, a few hundred kilobytes of SRAM, clock frequencies of less than a few hundred megahertz, and power consumption in the milliwatt range [3]. To deploy ML under these conditions, models must be compressed and there must be dedicated software packages that are able to convert, optimize and execute the models efficiently on the device [4].

In this paper, we will summarize the several frameworks that are available to deploy ML models to low-power MCUs and then compare their merits and practical properties. The objectives are to: (i) explain the hardware platforms and the model compression techniques that form the basis of TinyML; (ii) outline the key deployment frameworks from a memory footprint, latency, energy, accuracy retention and portability perspective; (iii) summarise the main sensing applications for IoT; and (iv) identify the key open research challenges that hamper the deployment of TinyML at scale.

2. Literature Review

2.1 Microcontroller Hardware for Edge Intelligence

TinyML aims at devices that span several orders of magnitude between mobile and server platforms currently used by common deep learning applications. The Arm Cortex-M series leads the way: Cortex-M4 and Cortex-M7 cores feature SIMD instructions and floating-point units; Cortex-M55 and Cortex-M85 cores bring Helium vector extensions; and can be combined with Ethos-U micro neural processing units (NPUs) for dedicated acceleration. The ESP32 family is used broadly when an integrated Wi-Fi and Bluetooth module is needed, and RISC-V based MCUs are getting noticed due to their open instruction set which enables the customization of ML extensions. In all these platforms, there are two limiting factors: (1) model weights and program code must be stored in flash memory, and (2) intermediate activations need to be stored in SRAM.

2.2 Model Compression and Optimisation Techniques

Most of the trained networks far surpasses the MCU memory hence model compression is a prerequisite for deployment. Post-training quantisation and quantisation-aware training quantises weights and activations from 32-bit floating point to 8-bit integers, which can save the model size by about 4x and perform integer-only operations without significant accuracy loss (typically no more than 1-2% error) [5]. Pruning eliminates redundant weights/weight propagation channels and with pruning, quantisation and weight sharing, the storage requirement can be significantly reduced [6].

An overview of the techniques in knowledge distillation and hardware-aware neural architecture search:An overview of knowledge distillation and hardware-aware neural architecture search techniques:

2.3 TinyML Software Frameworks

There are three generic deployment frameworks and they are mapped to a trained model to code for the MCU. Interpreter based runtime models include TF Lite for Microcontrollers (TFLM, now included in LiteRT), which runs a serialised model using a small, portable interpreter [7]. Other frameworks typically use hand-optimised operators for neural networks provided by kernel libraries, the most popular of which are CMSIS-NN for Arm Cortex-M processors [8]. Compiler-based tools and code generation tools (including vendor tools like STM32Cube) Remove interpreter overhead, memory can be scheduled ahead of time, AI (now part of ST Edge AI), the Apache TVM compiler stack [9], and the TinyEngine inference engine built with MCUNet – generate model-specific C code.

These are all wrapped up in an end-to-end platform like Edge Impulse, which includes data collection, training, optimisation, and firmware export [10].

2.4 IoT Sensing Applications

The TinyML project is most well established on a limited number of applications in which sensors are always on. Compact CNNs or depthwise-separable networks on microphone data are used for keyword spotting and audio event detection; for industrial machines, vibration and current signatures are used for anomaly detection and predictive maintenance; for wearables, low-resolution cameras are used for person detection and visual wake words; and for keyword spotting and audio event detection, there exists a class of networks based on long short-term memory (LSTM) which are trained on long sequences of audio data.

There is a class of networks based on the long short-term memory (LSTM) that operate on long sequences of audio data and are trained for keyword spotting and audio event detection. The areas in which it is used are expanding, such as environmental monitoring, smart agriculture, and health monitoring [4]. All the TinyML deployment stack (hardware platforms, optimisation techniques, deployment frameworks) are summarized in Table 1.

Table 1. The TinyML deployment stack for IoT sensing, linking microcontroller hardware platforms, model-optimisation techniques and software deployment frameworks.

TinyML Deployment Stack for IoT Sensing		
Hardware Platforms	**Model Optimisation**	**Deployment Frameworks**
Arm Cortex-M, ESP32, RISC-V MCUs (with optional Ethos-U micro-NPU)	Quantisation (int8) Pruning and knowledge distillation Neural architecture search (MCUNet)	TFLM/LiteRT, CMSIS-NN, ST Edge AI Edge Impulse, Apache TVM, TinyEngine

2.5 Energy, Latency and Benchmarking

In TinyML, comparing the performance of systems is challenging due to the dependency of the model, the system's framework, and compiler settings and hardware. Early studies were reported for custom models and boards, and cannot be compared across studies. To solve this problem, MLPerf Tiny has been introduced to establish four reference tasks that use standard models, datasets and quality metrics, and to evaluate latency and energy consumption using a consistent methodology [3]. The results submitted demonstrate that performance can differ significantly between similar software stacks on similar hardware for the same task, establishing that the stack is a first-order factor in TinyML performance.

2.6 Research Gaps

Although there have been some great strides, there are some common gaps that remain in the literature. First, a lot of studies still present results on custom models and boards, and not on common standards, which makes it hard to reproduce and compare. Second, the majority of frameworks provide only inference capabilities; on-device training and adaptation—where models learn from the drift in the sensors and changing conditions—in which they adapt themselves to the changes in the real world are still at an early stage. Third, there are different sets of vendors, toolchains and operators, and what works on one platform may require significant rework for another. Finally, aggressive compression can impact accuracy in a manner that is overlooked, especially for minority classes in the case of imbalanced sensing data [1], [4]. This review highlights the evidence that has been collected on these gaps and urges that evaluation using benchmarks should be a priority for future work, specifically deployment aware evaluation.

3. Materials and Methods

3.1 Review Scope and Method

The structured review process is used to search for, classify and compare deployment strategies for low power MCUs. The reviewed studies and tools were assigned into three categories: (i) execution model (interpreter-based runtime, kernel library, compiler-based code generation or end-to-end platform); (ii) target hardware (general Cortex M, vendor-specific MCU, RISC-V or NPU-accelerated device); and (iii) application workload (audio, vibration and inertial, or vision). For each category, the performance reported was recorded and the model features and deployment constraints were documented to enable a like-for-like comparison. Each framework was analysed as it is used in practice, such as in combination with the CMSIS-NN optimised kernels as is often done on Arm devices. For studies that provided results for multiple boards or models, each was considered a separate measurement and hardware-dependent trends were explored instead of being averaged across models.

3.2 Evaluation Criteria

The IoT frameworks are evaluated on six criteria pertinent to IoT deployment: memory footprint – peak SRAM and flash usage including the runtime; inference latency; energy per

inference; accuracy retention after quantisation and other optimisations; hardware portability; and ease of use – tooling and documentation. The above criteria are not only accuracy requirements, but also practical considerations covered in Section 2.6, as a highly accurate model that is too large to fit in memory, or too heavy on battery, cannot be deployed.

Additionally, all frameworks were evaluated qualitatively for licensing, operator support (the number of neural-network layers that the framework supports), and integration effort (the engineering effort required to integrate the model into the current firmware and real-time operating systems). Though not so easily measured, these are important influences on the likelihood that a lab prototype will become a field product.

3.3 Hardware Platforms and Benchmark Workloads

Results are grouped per every hardware class for comparability reasons: mid-range Cortex-M4/M7 MCUs, low-end Cortex-M0+/M3 devices and NPU-accelerated platforms. Where possible the workloads were matched to those of the Tiny reference tasks from the MLPerf benchmark, which offer models and quality targets [3]. Reported results on custom models were also included, but taken with a grain of salt, as differences in model architecture may outweigh differences between frameworks.

3.4 Deployment Workflow

The typical deployment procedure for the reviewed studies is as follows: (1) sensor data collection and pre-processing, (2) model training on a desktop platform (e.g., TensorFlow or PyTorch), (3) quantisation, pruning or model architecture search, (4) model conversion to a framework-specific format or C code, and (5) profiling of accuracy, latency, energy consumption on the target device. The accuracy should be validated on the device, not just in simulation; there may be slight differences from the desktop model because of integer arithmetic and implementation of operators in the device, and the memory reports should include the runtime and working buffers along with the model weights.

4. Results and Discussion

4.1 Comparison of Frameworks

The principal features of the main frameworks discussed are summarized in Table 2, which contrasts their execution model, hardware support, memory overhead and main strengths. The comparison is qualitative and based on representative studies for each of the frameworks, since, as previously explained (in Section 2.6), there is no body of studies that test all frameworks simultaneously under the same conditions.

Table 2. Comparison of software frameworks for deploying machine-learning models on low-power microcontrollers.

Framework	Execution model	Hardware support	Memory overhead	Key strength
TFLM / LiteRT	Interpreter runtime	Broad (Cortex-M, ESP32, RISC-V)	Low–moderate	Portable, large community

Framework	Execution model	Hardware support	Memory overhead	Key strength
CMSIS-NN	Kernel library	Arm Cortex-M	Very low	Fast optimised kernels
ST Edge AI (STM32Cube.AI)	Code generation	STM32 MCUs	Low	Vendor-tuned, memory reports
Edge Impulse	End-to-end platform	Broad, many development boards	Low–moderate	Rapid data-to-firmware workflow
MCUNet / TinyEngine	Co-design compiler	Arm Cortex-M	Very low	Fits larger models in SRAM

There are some patterns to the data in Table 2. The lowest memory overhead and latency is achieved by frameworks that produce specific code for the model or use a kernel for the specific hardware, because they do not require a general interpreter or plan memory usage before deployment [11]. The downside of portability is that vendor tools must be built for one MCU family, while interpreter-based runtimes like TFLM run on a variety of devices with relatively light overhead [7]. End-to-end platforms are built for faster development time in exchange for some lower-level control, which is appealing to smaller teams and fast prototyping [10].

4.2 Discussion

Three consistent findings emerge: The first is that there is no universal best framework; the best choice depends on the goal, whether it's minimum latency and memory on a known device, portability across hardware, or speed of development. Second, 8-bit integer quantisation is the most effective single optimisation, typically shrinking the model size by a factor of approximately four, while losing accuracy no more than 25%, and is compatible with all the frameworks reviewed [5]. Thirdly and most crucially, performance data from various studies is not directly comparable as they differ in the models, board types, clock speeds and measurement techniques they are based on (which is why it is important to have common benchmarks). Total energy per inference and reliable operation in memory are more important than peak accuracy on a single dataset for IoT deployments.

The findings also call into doubt the current reporting of TinyML performance. It is important to note that without any energy data included in latency statistics (or without the runtime and activation buffers in memory statistics) then the figures might be misleading with regard to deployability. The most informative measure of real world readiness is therefore benchmark-based evaluation, which is used to measure accuracy, latency and energy together on standard tasks, in this case MLPerf Tiny [3]. In this aspect, the distinctions among software stacks are more critical, and the selection of framework should not be overlooked.

5. Conclusion

Key frameworks for low-power deployment of machine learning models on microcontrollers powering the Internet of Things (IoT) sensing devices have been reviewed and compared. The evidence demonstrates that portable interpreter-based runtimes have progressed over the years

to optimised kernel libraries, vendor code generators and hardware–software co-design techniques, all backed by end-to-end platforms that streamline the development process. Where portability and community support are of great concern, interpreter-based frameworks are useful, and where memory and latency is most important, compiler-based and co-designed frameworks work best on target hardware.

When it comes to large-scale deployments of IoT, however, there are other factors that outweigh the comparison of accuracy: memory footprint, energy per inference, portability and maintainability. A more promising direction is thus the hardware–software co-design combining compressed models, the optimisation of their runtimes, and the development of emerging micro-NPUs. Addressing structural gaps in the field such as standardised benchmarking, on-device and federated learning, and interoperability between toolchains and transparent reporting of accuracy loss after compression will spur further progress. More attention should be paid to these enablers and more value should be placed on long-term field deployments as the criteria for assessing the frameworks.

References

[1] R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-enabled frugal smart objects: Challenges and opportunities," IEEE Circuits and Systems Magazine, vol. 20, no. 3, pp. 4-18, 2020.

[2] P. Warden and D. Situnayake, TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. O'Reilly Media, 2019.

[3] C. Banbury et al., "MLPerf Tiny benchmark," in Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, vol. 1, 2021.

[4] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 4, pp. 1595-1623, 2022.

[5] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2704-2713.

[6] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in International Conference on Learning Representations (ICLR), 2016.

[7] R. David et al., "TensorFlow Lite Micro: Embedded machine learning for TinyML systems," Proceedings of Machine Learning and Systems, vol. 3, pp. 800-811, 2021.

[8] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs," arXiv preprint arXiv:1801.06601, 2018.

[9] T. Chen et al., "TVM: An automated end-to-end optimizing compiler for deep learning," in Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018, pp. 578-594.

[10] S. Hymel et al., "Edge Impulse: An MLOps platform for tiny machine learning," Proceedings of Machine Learning and Systems, vol. 5, 2023.

[11] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," Advances in Neural Information Processing Systems, vol. 33, pp. 11711-11722, 2020.