INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)
(Int Peer Reviewed Journal)
Vol. 05, Issue 10, October 2025, pp : 405-408

www.ijprems.com
editor@ijprems.com

e-ISSN : 2583-1062
Impact Factor : 7.001

A COMPREHENSIVE STUDY ON TINYML: RUNNING MACHINE LEARNING MODELS ON MICROCONTROLLERS

Jain Mannat Nitin[1], Patel Uday Bhadreshkumar[2], Patel Jaimil Ravindrakumar[3], Patel Jaysmin Dilipbhai[4], Patel Janki Tejas[5]

[1,2,3,4,5]Department of Computer Engineering, SAL College of Engineering, Ahmedabad, India.

ABSTRACT

Tiny Machine Learning (TinyML) is an emerging discipline that enables the execution of machine learning inference on ultra-low-power, memory-constrained microcontrollers and edge devices. By shifting intelligence to the device, TinyML reduces latency, improves privacy, and lowers bandwidth and energy costs associated with cloud-centred architectures. This paper reviews the technical architecture and toolchain of TinyML, surveys prominent applications across domains, discusses model optimization and deployment strategies, and analyses challenges including resource constraints, energy management, security vulnerabilities, and update mechanisms. We also highlight promising research directions such as on-device learning, federated approaches, neural architecture search tailored for microcontrollers, and hardware–software co-design.

Keywords: TinyML; Edge AI; Microcontroller; TensorFlow Lite Micro; Quantization; On-device Inference.

1. INTRODUCTION

The proliferation of the Internet of Things (IoT) has produced a vast ecosystem of battery-powered and always-on devices that continuously collect sensor data. Historically, sensor streams have been transmitted to centralized servers or cloud platforms for processing and machine learning inference.

Although cloud-based solutions provide considerable compute resources, they introduce several drawbacks: higher end-to-end latency, dependence on network connectivity, increased operational cost, and concerns regarding user privacy and data sovereignty. TinyML addresses these constraints by enabling meaningful inference directly on microcontrollers (MCUs), which typically offer only tens to hundreds of kilobytes of RAM and a few megabytes of flash storage.

1.1 Background and Motivation

Edge intelligence and TinyML have emerged as an intersection of embedded systems and machine learning with the specific aim of delivering real-time, privacy-preserving analytics in resource-constrained environments.

Motivations driving TinyML adoption include the need for ultra-low latency for safety-critical applications, reduced communication energy for distributed sensor networks, and compliance with privacy regulations by minimizing raw data transmission. Additionally, advances in software toolchains and compiler optimizations have lowered the barrier to porting compact models to MCUs, enabling practical deployment scenarios in remote and industrial settings.

1.2 Contributions of this Paper

This paper retains the core exposition of TinyML while expanding the technical depth on optimization techniques, benchmarks, and practical deployment considerations. The principal contributions are: (i) a systematic overview of the TinyML toolchain and optimization strategies; (ii) a comparative summary of frameworks and hardware platforms suited for TinyML; (iii) an analysis of measurement metrics and benchmarking practices; and (iv) a discussion of research directions including federated and continual learning on constrained devices.

2. TECHNICAL ARCHITECTURE OF TINYML

A standard TinyML pipeline begins with data collection and offline training on a workstation or cloud instance, proceeds through model compression and conversion to an inference-ready format, and culminates in deployment to an MCU where an optimized runtime performs inference. Figure 1 illustrates a canonical TinyML system architecture, highlighting the development and deployment stages.

2.1 Model Optimization Techniques

Model optimization is central to TinyML. Popular techniques include: - Quantization: Converting 32-bit floating-point weights and activations to 8-bit integers (or lower) reduces model size and enables faster integer-only inference on MCUs without an FPU. Per-channel quantization, symmetric/asymmetric schemes, and proper calibration via a representative dataset are crucial to reduce accuracy.

degradation. - Pruning and sparsity: Removing redundant weights or entire channels and leveraging sparsity-aware runtimes can reduce memory and compute costs. However, sparse representations may complicate runtime implementation on simple MCUs. - Knowledge distillation: Training a smaller 'student' model to mimic a larger 'teacher' model yields compact networks that retain accuracy. – Architecture search and micro-design: Manual or automated neural architecture search (NAS) tailored for MCU constraints can yield highly efficient microarchitectures (e.g., depthwise separable convolutions, pointwise layers).

2.2 Inference Engines and Conversion Toolchains

A robust toolchain reduces friction when porting models to constrained hardware. TensorFlow Lite for Microcontrollers (TFLM) provides a minimal runtime with C++ APIs and example projects (e.g., keyword spotting) and supports integer-only models. CMSIS-NN offers highly optimized kernels for ARM Cortex-M processors, improving execution efficiency. Edge Impulse and platform-specific exporters provide convenient pipelines from data collection to an MCU-flashable binary. Compiler toolchains and link-time optimizations also play a role in minimizing binary size.

3. APPLICATIONS OF TINYML

TinyML's low-power, on-device inference enables a wide variety of applications across domains. Below we detail representative use cases and practical considerations for deployment.

3.1 Healthcare and Wearables

In healthcare, TinyML enables continuous monitoring for arrhythmia, sleep apnea, and fall detection directly on wearable devices. Privacy is a critical requirement; hence inference on-device prevents raw biometric signals from leaving the user's device. Low false-positive rates and explainability are important in medical settings; therefore models must be validated across diverse demographics and environmental conditions.

3.2 Agriculture and Environmental Monitoring

Agricultural deployments benefit from TinyML when connectivity is intermittent. Models that classify crop disease from camera images, detect pest activity via acoustic signatures, or predict irrigation needs from

multi-sensor fusion can operate autonomously in the field, reducing the need for costly data transmission. Energy harvesting and low-power radios extend the operational lifetime of such devices.

3.3 Industrial IoT and Predictive Maintenance

TinyML enables anomaly detection and predictive maintenance by continuously analysing vibration, temperature, and acoustic sensors. Detecting subtle anomalies at the edge allows immediate local mitigation actions and reduces the frequency of costly machine downtime.

4. TOOLS, FRAMEWORKS, AND HARDWARE PLATFORMS

This section compares common frameworks and hardware choices. TensorFlow Lite Micro is widely adopted due to its active community and

and one-click deployment, suitable for rapid prototyping. CMSIS-NN is indispensable for ARM Cortex-M devices, delivering near-optimal performance for convolutional and fully-connected layers.

4.1 Hardware Comparison

Common hardware targets include ARM Cortex-M series (M0/M3/M4/M7), which provide a balance of performance and low power; ESP32, which includes Wi-Fi and dual-core processors for heavier workloads; and specialized boards like Arduino Nano 33 BLE Sense that integrate sensors for rapid prototyping. STM32 microcontrollers offer a wide range of memory and peripheral options and are often selected for industrial-grade TinyML deployments.

Platform	Typical RAM	Flash	Notes
ARM Cortex-M0/M3	16–64 KB	128–512KB	Extremely Low-Powered, Limited compute
ARM Cortex-M4/M7	64–512 KB	1–4 MB	DSP Extensions, FPU on some variants
ESP32	320 KB (approx)	4 MB (typ)	WiFi/Bluetooth Higher power
Arduino Nano 33 BLE Sense	64 KB	1 MB	Onboard Sensors, Good for prototyping

5. CHALLENGES AND LIMITATIONS

Despite its promise, TinyML faces several technical and operational challenges that constrain adoption in critical domains. Below we examine these limitations in detail.

5.1 Memory and Computation Constraints

Typical microcontrollers provide severely constrained RAM and flash. Memory fragmentation, stack/heap usage by runtimes, and the need to load model weights into flash (or external memory) require careful engineering.

Developers must minimize dynamic allocations and rely on static buffers where possible.

5.2 Energy and Power Management

Battery-operated TinyML devices must balance inference frequency with energy consumption. Aggressive duty-cycling, event-driven sensing, and ultra-low-power wake-up mechanisms are key design patterns. When continuous sampling is required, careful sensor selection and low-power front-ends help extend lifetime.

5.3 Security, Privacy, and Maintainability

Local inference reduces transfer of raw data but introduces firmware and model integrity concerns. Models and firmware must be signed and protected to prevent tampering. Furthermore, side-channel attacks and model extraction threats have been demonstrated on embedded ML stacks, necessitating countermeasures such as masking, obfuscation, and secure boot. Maintainability and secure update mechanisms (OTA) are essential for long-lived deployments.

5.4 Model Lifecycle and On-device Adaptation

Training remains costly and typically offline; thus, in-field model updates are often performed by replacing firmware images. Emerging research on

on-device incremental learning, federated learning, and efficient personalization techniques aims to reduce reliance on full re-training cycles while preserving privacy and computational feasibility.

6. FUTURE SCOPE AND RESEARCH DIRECTIONS

TinyML research is rapidly evolving. The following directions offer promising opportunities for improving both capability and reliability of on-device intelligence.

6.1 Federated and On-device Learning

Federated learning enables model improvements by aggregating weight updates from client devices without transferring raw data. Research into communication-efficient update protocols and compression of model gradients is necessary to make federated approaches viable on severely constrained hardware. On-device continual learning methods that avoid catastrophic forgetting and require minimal compute remain an open research challenge.

6.2 Neural Architecture Search & AutoML for TinyML Automated methods to discover microarchitectures optimized for MCU constraints can provide significant efficiency gains. NAS techniques that incorporate energy and memory constraints into objective functions are particularly relevant for TinyML.

6.3 Hardware–Software Co-design

Co-design of lightweight models and microcontroller features — including vector extensions, optimized instruction sets, and small accelerators — will allow more expressive models to execute efficiently at the edge. The emergence of low-power NPUs and RISC-V-based microcontrollers with vector extensions promises new optimization opportunities.

6.4 Sustainability and Green AI

TinyML naturally aligns with green computing objectives due to reduced data transfer and lower energy per inference. Evaluating environmental impact using standardized metrics and designing models that minimize carbon footprint are meaningful directions for future work.

7. EXPERIMENTAL GUIDELINES AND METRICS

Researchers should evaluate TinyML systems using multiple orthogonal metrics: model size (bytes), peak RAM usage, latency per inference (ms), energy per inference (mJ), throughput (inferences per second), and standard accuracy metrics (accuracy, F1 score). When possible, adopt community benchmarks such as MLPerf Tiny to enable reproducible comparisons. Power should be measured using instrumentation (e.g., Monsoon, Otii Arc) or

high-resolution shunt measurement. Memory profiling and linker map analysis help ensure that models and runtimes meet target constraints.

8. CONCLUSION

This paper presents an expanded overview of TinyML, describing the architecture, optimization techniques, application domains, implementation tools, and practical challenges. TinyML reduces latency and improves privacy by enabling on-device inference, yet significant research remains to address constraints in memory, energy, security and maintainability. Continued progress in toolchains, model optimization, and hardware advances will accelerate adoption across commercial and societal applications.

9. REFERENCES

[1] M. Shafique, T. Theocharides, V. Reddi, and B. Murmann, “TinyML: Current Progress, Research Challenges, and Future Roadmap,” IEEE Design Automation Conference, 2021.

[2] Y. Abadade et al., “A Comprehensive Survey on TinyML,” IEEE Access, vol. 11, 2023.

[3] V. Janapa Reddi et al., “MLPerf Tiny Benchmark,” NeurIPS Datasets and Benchmarks Track, 2021.

[4] J. Lin et al., “Tiny machine learning: Progress and futures,” IEEE Circuits and Systems Magazine, 2023.

[5] P. P. Ray, “A review on TinyML: State-of-the-art and prospects,” Journal of King Saud University - Computer and Information Sciences, 2022.

[6] C. Banbury et al., “Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers,” MLSys, 2021.

[7] Edge Impulse documentation and TensorFlow Lite for Microcontrollers resources.

[8] CMSIS-NN: ARM optimized neural network kernels for Cortex-M devices.