International Journal of Research and Review Techniques (IJRRT), ISSN: 3006-1075
Volume 5, Issue 1, January-March, 2026, Available online at: https://ijrrt.com

Edge AI and TinyML: Optimizing Inference on Low-Power Microcontrollers

Vijay Janapa Reddi

Associate Professor, John A. Paulson School of Engineering and Applied Sciences, Harvard University, USA

ABSTRACT

The rapid advancement of artificial intelligence (AI) has created a growing demand for intelligent systems capable of performing real-time inference with minimal computational resources and energy consumption. Edge AI and Tiny Machine Learning (TinyML) address this challenge by enabling machine learning models to operate directly on resource-constrained devices, such as low-power microcontrollers, without relying on continuous cloud connectivity. This paper explores optimization techniques for deploying and executing machine learning models on microcontrollers with limited memory, processing capability, and power availability. Key approaches include model quantization, pruning, knowledge distillation, operator optimization, and efficient memory management to reduce model size, computational complexity, and inference latency. Furthermore, the study examines the trade-offs between inference accuracy, energy efficiency, execution speed, and memory utilization across TinyML applications, including environmental monitoring, wearable healthcare, industrial automation, and smart sensing. The role of hardware-aware optimization and lightweight inference frameworks in improving deployment efficiency is also discussed. By integrating algorithmic optimization with hardware-level considerations, Edge AI and TinyML can enable reliable, low-latency, and energy-efficient intelligence at the network edge. This paper highlights the challenges and opportunities in developing scalable TinyML solutions and emphasizes the importance of balancing predictive performance with the strict resource constraints of low-power microcontrollers.

Keywords: Edge AI, TinyML, low-power microcontrollers, model quantization, model pruning, inference optimization, energy efficiency, embedded machine learning.

INTRODUCTION

The rapid evolution of Artificial Intelligence (AI) and the Internet of Things (IoT) has significantly increased the demand for intelligent, energy-efficient, and real-time computing systems. Traditional AI applications rely heavily on cloud computing for data processing and model inference. However, cloud-based approaches often introduce communication latency, require continuous network connectivity, increase bandwidth consumption, and raise concerns regarding data privacy and security. To overcome these limitations, Edge AI has emerged as a promising approach that enables data processing and intelligent decision-making directly on edge devices, closer to the source of data generation.

Tiny Machine Learning (TinyML) is a specialized branch of Edge AI that focuses on deploying machine learning models on resource-constrained embedded devices, particularly low-power microcontrollers (MCUs). These devices typically operate with limited memory, processing capability, and energy resources, making the deployment of conventional machine learning models challenging. Despite these constraints, TinyML enables applications such as predictive maintenance, wearable healthcare monitoring, environmental sensing, smart agriculture, voice recognition, and industrial automation. By performing inference locally, TinyML reduces dependence on cloud infrastructure and supports low-latency operation with improved energy efficiency.

However, deploying machine learning models on low-power microcontrollers presents several technical challenges. The limited availability of random-access memory (RAM), flash storage, and computational resources restricts the size and complexity of models that can be executed efficiently. Furthermore, continuous inference may increase energy consumption, while aggressive optimization can negatively affect model accuracy. Therefore, achieving an appropriate balance among inference accuracy, latency, memory utilization, and power consumption is a critical requirement for practical TinyML deployment.

To address these challenges, several optimization techniques have been developed, including model quantization, pruning, knowledge distillation, operator-level optimization, and efficient memory management. Quantization reduces the numerical precision of model parameters and computations, thereby decreasing memory requirements and computational overhead. Pruning removes redundant model parameters or connections to reduce model complexity, while knowledge distillation transfers knowledge from a larger model to a smaller, more efficient model. Additionally,

hardware-aware optimization and specialized inference frameworks help improve execution speed and resource utilization on microcontrollers.

The effectiveness of these techniques depends on the characteristics of the machine learning model, the target hardware architecture, and the requirements of the intended application. For instance, applications involving wearable health monitoring may prioritize low power consumption and reliable predictions, whereas industrial monitoring systems may emphasize low inference latency and consistent real-time performance. Consequently, optimization strategies must be selected according to application-specific requirements and hardware limitations.

This paper focuses on optimizing machine learning inference on low-power microcontrollers using Edge AI and TinyML techniques. It examines the major challenges associated with resource-constrained inference and discusses methods for reducing model size, computational complexity, memory usage, and energy consumption while maintaining acceptable prediction accuracy. The paper also highlights the importance of evaluating inference latency, memory footprint, energy efficiency, and model performance when selecting suitable optimization techniques. Through this analysis, the study aims to provide insights into the development of efficient, reliable, and scalable TinyML systems for next-generation embedded intelligence.

2. Proposed Models and Methodologies

The proposed methodology focuses on optimizing machine learning inference on low-power microcontrollers using Edge AI and TinyML techniques. The primary objective is to reduce computational complexity, memory consumption, inference latency, and energy usage while maintaining acceptable model accuracy. The proposed framework integrates lightweight model architectures, model compression techniques, hardware-aware optimization, and efficient deployment strategies to enable real-time inference on resource-constrained embedded devices.

2.1 Proposed System Architecture

The proposed Edge AI and TinyML system consists of five major stages: data acquisition, preprocessing, model development, model optimization, and deployment on a low-power microcontroller.

1. Data Acquisition: Sensor data are collected from embedded devices, such as temperature sensors, accelerometers, microphones, or environmental monitoring systems. These data serve as input for training and evaluating the machine learning model.

2. Data Preprocessing: The collected data are cleaned, normalized, and transformed into a suitable format. Feature extraction and dimensionality reduction may be applied to minimize unnecessary computations and improve model efficiency.

3. Model Development: A lightweight machine learning or deep learning model is selected according to the application's requirements. Suitable architectures include small convolutional neural networks (CNNs), compact multilayer perceptrons (MLPs), and decision-tree-based models.

4. Model Optimization: The trained model is optimized using quantization, pruning, knowledge distillation, and operator fusion, where supported. These techniques reduce model size and computational requirements while attempting to preserve prediction accuracy.

5. Deployment and Evaluation: The optimized model is converted into a microcontroller-compatible format and deployed using an embedded inference framework, such as TensorFlow Lite for Microcontrollers. Performance is evaluated using inference latency, memory footprint, energy consumption, and predictive accuracy.

2.2 Proposed Machine Learning Models

The proposed methodology considers lightweight models that can operate within the memory and processing constraints of microcontrollers.

A. Lightweight Convolutional Neural Network (CNN)

A compact CNN is suitable for applications involving time-series sensor signals, vibration patterns, audio features, and image data. By using a limited number of convolutional layers and filters, the model can extract relevant features while reducing computational overhead. Depthwise separable convolutions may further reduce the number of parameters and multiply-accumulate operations compared with conventional convolutional layers.

B. Multilayer Perceptron (MLP)

An MLP is appropriate for structured sensor data and applications involving a relatively small number of input features. Its simple architecture makes it suitable for microcontrollers with limited RAM and processing capacity. The number of hidden layers and neurons can be adjusted to achieve a balance between predictive accuracy and computational efficiency.

C. Decision Tree and Compact Classical Models
Decision trees and other lightweight classical machine learning models can be used for classification and regression tasks involving structured data. Their relatively low computational requirements make them useful for applications where memory efficiency and fast inference are more important than learning complex representations.

The final model should be selected according to the application, dataset, hardware specifications, and acceptable accuracy requirements. A compact CNN or MLP may be used as the primary model for evaluating the effects of TinyML optimization techniques.

2.3 Model Optimization Techniques
The proposed framework applies several optimization techniques to reduce the resource requirements of machine learning inference.

A. Model Quantization
Quantization converts model parameters and, where supported, intermediate computations from higher-precision representations, such as 32-bit floating point, to lower-precision formats, such as 8-bit integers. Integer quantization can substantially reduce model storage requirements and improve inference efficiency on hardware with suitable integer arithmetic support.

For example, reducing model weights from 32-bit floating point to 8-bit integers reduces the theoretical storage required for those weights by approximately 75%, excluding additional metadata and implementation overhead. The impact on inference accuracy must be evaluated after quantization.

B. Model Pruning
Pruning removes redundant weights, connections, or structures from a trained model to reduce its complexity. Unstructured pruning removes individual weights, whereas structured pruning removes entire filters, channels, or other model components. Structured pruning is particularly useful when the target inference engine can exploit the resulting smaller architecture. The effectiveness of pruning depends on whether the hardware and software implementation can translate the reduced model complexity into actual memory or latency savings.

C. Knowledge Distillation
Knowledge distillation transfers learned information from a larger teacher model to a smaller student model. The student model is trained to approximate the teacher's predictions while using fewer parameters and fewer computational operations. This technique can help maintain predictive performance when deploying compact models on resource-constrained microcontrollers.

D. Operator and Memory Optimization
Operator optimization involves selecting efficient implementations of mathematical operations supported by the target microcontroller. Hardware-specific libraries can accelerate operations such as matrix multiplication, convolution, and activation functions. In addition, memory optimization techniques, including buffer reuse, tensor memory planning, and removal of unnecessary intermediate data, help reduce peak RAM usage during inference.

2.4 Proposed Optimization Workflow
The proposed methodology follows an iterative workflow in which model performance is measured before and after optimization.

Data Collection and Preprocessing
Sensor data acquisition, normalization, and feature extraction

Baseline Model Training
Train and evaluate a compact CNN, MLP, or classical model

Model Optimization
Quantization, pruning, distillation, and memory optimization

Microcontroller Deployment
Convert and deploy using a compatible embedded inference runtime

Performance Evaluation
Measure accuracy, latency, RAM, flash usage, and energy per inference

Figure 1. Proposed methodology for optimizing TinyML inference on low-power microcontrollers.

Initially, a baseline model is trained and evaluated using a predefined dataset. The baseline measurements establish reference values for predictive accuracy, model size, and computational requirements. Subsequently, optimization techniques are applied individually and in combination. The optimized models are then deployed on the target microcontroller and evaluated under consistent operating conditions.

The process is repeated until a suitable balance between resource consumption and predictive performance is achieved. If the optimized model exceeds the available memory budget or fails to satisfy latency requirements, the model architecture and optimization settings are revised.

2.5 Performance Evaluation Metrics

The effectiveness of the proposed methodology is assessed using the following metrics.

Metric	Purpose
Model accuracy (%)	Measures the correctness of model predictions.
Inference latency (ms)	Measures the time required to generate a prediction.
RAM usage (KB)	Measures runtime memory requirements.
Flash usage (KB)	Measures storage required for the deployed model and related program components.
Energy per inference (mJ)	Measures the energy required to complete one inference.
Model size (KB)	Measures the storage occupied by model parameters and associated data.

For classification tasks, accuracy can be calculated as:
Accuracy=TP+TNTP+TN+FP+FN×100\text{Accuracy}=\frac{TP+TN}{TP+TN+FP+FN}\times100
where TPTP, TNTN, FPFP, and FNFN represent true positives, true negatives, false positives, and false negatives, respectively.

The energy consumed per inference can be estimated as:
Einf=Pavg×TinfE_{\text{inf}}=P_{\text{avg}}\times T_{\text{inf}}
where PavgP_{\text{avg}} is the average power consumed during inference and TinfT_{\text{inf}} is the inference duration. When power is expressed in milliwatts and time in milliseconds, the resulting energy is in microjoules; conversion to millijoules requires dividing by 1,000.

A fair evaluation requires consistent input data, hardware configuration, clock frequency, measurement conditions, and inference settings across the baseline and optimized models.

2.6 Expected Outcomes

The proposed methodology is expected to demonstrate that combining lightweight model architectures with quantization, pruning, and efficient memory management can improve the feasibility of deploying machine learning models on low-power microcontrollers. Potential benefits include reduced model storage, lower RAM requirements, improved inference speed, and reduced energy consumption. However, the magnitude of these improvements depends on the selected model, target hardware, inference framework, and optimization strategy.

The final objective is to identify an effective optimization configuration that satisfies the application's accuracy requirements while remaining within the available memory, latency, and energy budgets. The proposed framework provides a systematic basis for comparing different TinyML models and optimization techniques and for developing efficient Edge AI systems for real-time embedded applications.

RESULTS AND ANALYSIS

The performance of the proposed Edge AI and TinyML methodology is evaluated in terms of model accuracy, inference latency, memory utilization, model size, and energy consumption. The primary objective is to analyze the effectiveness of model optimization techniques in enabling efficient machine learning inference on low-power microcontrollers. The evaluation compares a baseline model with optimized models using quantization, pruning, and combined optimization techniques.

Note: Since no experimental dataset or microcontroller measurements have been provided, the following analysis presents an illustrative results framework. The numerical values are hypothetical examples for academic presentation and should be replaced with actual experimental measurements before submission as research findings.

3.1 Performance Comparison of Baseline and Optimized Models

The baseline model represents a machine learning model deployed without additional compression or hardware-specific optimization. The optimized models incorporate different techniques to reduce resource requirements and improve inference efficiency.

The following table illustrates a possible comparison of model performance.

Table 1. Illustrative comparison of TinyML optimization techniques. These values are hypothetical and do not represent measured experimental results.

Evaluation Metric	Baseline Model	INT8 Quantization	Pruning	Combined Optimization
Model size (KB)	256	68	180	52
RAM usage (KB)	96	54	82	42
Inference latency (ms)	45	24	38	19
Energy per inference (mJ)	2.70	1.44	2.28	1.14
Accuracy (%)	96.2	95.7	95.4	95.1

The illustrative results suggest that optimization can substantially reduce the resource requirements of machine learning models. INT8 quantization reduces the model's numerical precision, which can decrease storage requirements and accelerate inference on microcontrollers with efficient integer arithmetic support. Pruning removes redundant model parameters, although the actual benefits depend on whether the inference runtime and hardware can exploit the resulting sparsity.

The combined optimization approach demonstrates the lowest illustrative model size, RAM usage, inference latency, and energy consumption. However, it also exhibits a small reduction in predictive accuracy compared with the baseline. This highlights the need to balance model efficiency against application-specific accuracy requirements.

3.2 Analysis of Model Size and Memory Utilization

Model size and RAM utilization are critical constraints in TinyML deployment. Microcontrollers generally provide considerably fewer computational and memory resources than conventional computers. Therefore, reducing the model's storage requirements is essential for deploying machine learning applications on resource-constrained devices.
Model Size Comparison

The analysis highlights the potential of Edge AI and TinyML optimization to enable practical machine learning inference on low-power microcontrollers. Reducing model size and computational requirements can make it possible to deploy intelligent functions on devices that cannot support conventional machine learning workloads. Local inference also reduces dependence on continuous network connectivity and may improve responsiveness and data privacy.

Nevertheless, the effectiveness of optimization depends on the relationship between the model architecture, inference framework, and hardware platform. A technique that improves model size may not proportionally improve latency or energy consumption. Therefore, comprehensive evaluation should include both software-level measurements and hardware-level profiling.

In conclusion, the proposed methodology provides a structured approach for investigating the trade-offs between model accuracy and computational efficiency in TinyML systems. Experimental validation on a real microcontroller, using a defined dataset and reproducible measurement procedures, is necessary to establish the actual performance gains and determine the most effective optimization strategy for a specific application.

COMPARATIVE ANALYSIS

The following table compares different TinyML optimization techniques based on model size, RAM usage, inference latency, energy consumption, accuracy, and overall efficiency.

Table 2. Comparative analysis of Edge AI and TinyML optimization techniques

Performance Metric	Baseline Model	INT8 Quantization	Model Pruning	Combined Optimization
Model Size (KB)	256	68	180	52
RAM Usage (KB)	96	54	82	42
Inference Latency (ms)	45	24	38	19
Energy per Inference (mJ)	2.70	1.44	2.28	1.14
Model Accuracy (%)	96.2	95.7	95.4	95.1
Model Size Reduction (%)	0	73.4	29.7	79.7
Latency Reduction (%)	0	46.7	15.6	57.8
Energy Reduction (%)	0	46.7	15.6	57.8

Note: All numerical values are illustrative and hypothetical, intended for comparative analysis. They must be replaced with experimentally measured results before being reported as research findings.

Comparison of Optimization Techniques

Technique	Main Advantage	Major Limitation	Suitable Application
INT8 Quantization	Reduces model storage and may accelerate integer inference	May introduce accuracy loss	Wearable devices and sensor classification
Model Pruning	Removes redundant model parameters	Speed gains depend on sparse computation support	Predictive maintenance and anomaly detection
Knowledge Distillation	Enables smaller models to approximate larger models	Requires teacher–student training	Complex classification on resource-constrained devices
Memory Optimization	Reduces peak RAM requirements	Does not necessarily reduce model parameter count	Microcontrollers with limited SRAM
Hardware-Aware Optimization	Improves execution on the target processor	May require platform-specific tuning	Real-time embedded systems
Combined Optimization	Can balance storage, latency, and energy requirements	Requires careful tuning to preserve accuracy	Battery-powered Edge AI applications

Key Findings

- Model size: Combined optimization achieves the largest illustrative reduction in storage requirements.
- Inference latency: INT8 quantization and combined optimization offer the greatest illustrative latency improvements.
- Energy efficiency: Reduced inference time may lower energy consumption, provided the power profile remains comparable.
- Accuracy: The baseline model achieves the highest illustrative accuracy, demonstrating the trade-off between compression and predictive performance.
- Deployment suitability: The best approach depends on the microcontroller's RAM, flash storage, processing capabilities, and application-specific accuracy requirements.

Overall, the comparative analysis suggests that combining quantization with carefully selected complementary optimization techniques is a promising strategy for efficient TinyML deployment. However, actual performance improvements must be validated on the target hardware under consistent experimental conditions.

SIGNIFICANCE OF THE TOPIC

Edge AI and TinyML: Optimizing Inference on Low-Power Microcontrollers is a significant research topic because it addresses the growing need for intelligent, energy-efficient, and real-time computing in resource-constrained environments. By enabling machine learning models to operate directly on embedded devices, TinyML reduces dependence on cloud computing and supports the development of autonomous, low-cost, and energy-efficient intelligent systems.

The major significance of this topic is outlined below.
1. Energy Efficiency: Optimization techniques such as quantization, pruning, and efficient memory management reduce computational requirements and can lower energy consumption, extending battery life in portable and remote devices.

2. Real-Time Processing: Local inference reduces the communication delays associated with sending data to cloud servers, enabling faster decision-making in time-sensitive applications such as industrial fault detection and wearable monitoring.

3. Reduced Memory and Computational Requirements: TinyML makes it possible to deploy machine learning models on microcontrollers with limited RAM, flash storage, and processing capabilities, expanding AI accessibility to low-cost embedded platforms.

4. Enhanced Data Privacy and Security: Processing sensitive information locally can reduce the need to transmit raw data over networks, helping minimize privacy risks. However, device-level security measures are still necessary.

5. Reduced Network Dependence: Edge-based intelligence can continue operating without continuous internet connectivity, making it suitable for remote agricultural monitoring, environmental sensing, and industrial applications.

6. Cost-Effective AI Deployment: Low-power microcontrollers are often less expensive and consume less energy than more powerful computing platforms, enabling economical deployment of intelligent devices at scale.

7. Wide Range of Applications: The technology supports applications in smart healthcare, precision agriculture, smart homes, industrial automation, environmental monitoring, and predictive maintenance.

8. Sustainable Computing: Reducing computation, communication, and energy requirements can contribute to more sustainable embedded AI systems, although the overall environmental benefit depends on the complete device lifecycle.

Research Significance

From a research perspective, this topic emphasizes the trade-off between model accuracy, inference latency, memory utilization, and energy consumption. Investigating these factors helps identify optimization strategies that satisfy the practical constraints of low-power microcontrollers without unacceptable degradation in predictive performance.

In conclusion, Edge AI and TinyML provide an important foundation for the next generation of intelligent embedded systems. Optimizing inference on low-power microcontrollers can make AI more accessible, responsive, energy-efficient, and practical for real-world applications, particularly where computing resources, network connectivity, and battery capacity are limited.

LIMITATIONS AND DRAWBACKS

Although Edge AI and TinyML enable efficient machine learning inference on low-power microcontrollers, several technical and practical limitations affect their performance, scalability, and deployment. These limitations arise primarily from restricted hardware resources, model optimization trade-offs, and the challenges of maintaining reliable inference in real-world environments.

1. Limited Memory and Storage: Low-power microcontrollers have restricted RAM and flash storage, which limit the size of machine learning models, intermediate tensors, and runtime libraries that can be deployed. Complex deep learning architectures may require substantial simplification before deployment.

2. Reduced Model Accuracy: Optimization techniques such as quantization and pruning can introduce numerical errors or remove important model parameters. These changes may reduce predictive accuracy, particularly for complex classification tasks or datasets with subtle patterns.

3. Limited Computational Capability: Microcontrollers generally have lower processing capabilities than CPUs, GPUs, and dedicated AI accelerators. Consequently, computationally intensive models may experience higher inference latency or may not meet strict real-time requirements.

4. Energy Constraints: Although TinyML can reduce energy consumption, continuous sensing and inference may still drain battery-powered devices. Power consumption also depends on sensor activity, memory access, processor frequency, and communication requirements.

5. Hardware Compatibility Issues: Different microcontrollers support different instruction sets, memory configurations, and acceleration features. A model optimized for one platform may not achieve equivalent performance on another, requiring hardware-specific adjustments.

6. Complexity of Model Optimization: Techniques such as quantization-aware training, structured pruning, and knowledge distillation require careful configuration and validation. Selecting suitable optimization parameters can increase development time and require specialized expertise.

7. Limited Support for Complex Models: Advanced architectures involving large neural networks, attention mechanisms, or extensive input data may exceed the memory and computational budgets of low-power devices. This restricts the use of TinyML in applications requiring highly complex reasoning or high-resolution data processing.

8. Difficulties in Model Updating: Updating deployed models can be challenging for devices operating in remote or disconnected environments. Firmware compatibility, limited storage, update bandwidth, and secure deployment procedures can further complicate maintenance.

9. Security and Privacy Risks: Local processing can reduce data transmission, but embedded devices remain vulnerable to physical tampering, model extraction, adversarial inputs, and insecure firmware. Additional security mechanisms may increase computational and memory overhead.

10. Limited Standardization and Evaluation: Differences in hardware, inference frameworks, datasets, measurement procedures, and power profiles make it difficult to compare TinyML optimization techniques fairly. Reliable comparisons require consistent evaluation conditions and reproducible experiments.

Summary of Major Limitations

Limitation	Primary Impact	Possible Mitigation
Limited RAM and flash	Restricts model size and complexity	Quantization and memory planning
Accuracy degradation	Reduces prediction reliability	Quantization-aware training and fine-tuning
Low processing capability	Increases inference latency	Lightweight architectures and optimized kernels
Battery constraints	Limits operational lifetime	Duty cycling and energy-aware inference
Hardware dependency	Reduces portability	Hardware-aware model design
Optimization complexity	Increases development effort	Automated optimization and profiling tools
Security vulnerabilities	Exposes devices to attacks	Secure boot, signed updates, and input validation
Difficult model updates	Complicates maintenance	Secure and efficient update mechanisms

Overall Discussion

The primary drawback of Edge AI and TinyML is the need to balance predictive accuracy against strict hardware and energy constraints. Aggressive optimization may reduce model size and inference time but can also compromise prediction quality. Similarly, a model that performs efficiently on one microcontroller may require further optimization for another platform.

Therefore, successful TinyML deployment requires careful model selection, hardware-aware optimization, systematic performance measurement, and security-conscious implementation. Addressing these limitations is essential for developing reliable, scalable, and energy-efficient embedded AI systems.

CONCLUSION

Edge AI and Tiny Machine Learning (TinyML) represent promising approaches for deploying intelligent applications on low-power microcontrollers with limited memory, computational capacity, and energy resources. This paper examined the importance of optimizing machine learning inference to achieve an effective balance between model accuracy, inference latency, memory utilization, and energy consumption. Techniques such as model quantization, pruning, knowledge distillation, efficient memory management, and hardware-aware optimization can reduce computational overhead and facilitate the deployment of lightweight machine learning models on resource-constrained embedded devices.

The comparative analysis highlights the potential benefits of combining multiple optimization techniques to reduce model size, accelerate inference, and improve energy efficiency. However, these improvements may involve trade-offs in predictive accuracy and depend on the selected model architecture, target hardware, and inference framework. Therefore, optimization strategies must be carefully selected and validated according to application-specific requirements. Reliable evaluation should consider not only software-level metrics but also actual hardware measurements under consistent operating conditions.

Despite challenges related to hardware limitations, model accuracy degradation, security, and deployment complexity, TinyML offers significant opportunities in wearable healthcare, smart agriculture, industrial automation, environmental monitoring, and predictive maintenance. Future research can focus on automated model optimization, hardware-aware neural architecture search, adaptive inference, ultra-low-power AI accelerators, and secure model deployment to further improve the efficiency and reliability of embedded intelligence.

REFERENCES

1. Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., & Hafid, A. S. (2023). A comprehensive survey on TinyML. *IEEE Access, 11*, 96892–96922. https://doi.org/10.1109/ACCESS.2023.3294111
2. Capogrosso, L., Cunico, F., Cheng, D. S., Fummi, F., & Cristani, M. (2024). A machine learning-oriented survey on tiny machine learning. *IEEE Access, 12*, 23406–23426. https://doi.org/10.1109/ACCESS.2024.3365349
3. Immonen, R., & Hämäläinen, T. (2022). Tiny machine learning for resource-constrained microcontrollers. *Journal of Sensors, 2022*, Article 7437023. https://doi.org/10.1155/2022/7437023
4. Heydari, S., & Mahmoud, Q. H. (2025). Tiny machine learning and on-device inference: A survey of applications, challenges, and future directions. *Sensors, 25*(10), Article 3191. https://doi.org/10.3390/s25103191
5. Tsoukas, V., Gkogkidis, A., Boumpa, E., & Kakarountas, A. (2024). A review on the emerging technology of TinyML. *ACM Computing Surveys, 56*, Article 259. https://doi.org/10.1145/3661820

6. David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T., & Warden, P. (2021). TensorFlow Lite Micro: Embedded machine learning on TinyML systems. *Proceedings of Machine Learning and Systems, 3*. https://arxiv.org/abs/2010.08678
7. Lin, J., Chen, W.-M., Lin, Y., Cohn, J., Gan, C., & Han, S. (2020). MCUNet: Tiny deep learning on IoT devices. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 11711–11722). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/hash/86c51678350f656dcc7f490a43946ee5-Abstract.html
8. Warden, P., & Situnayake, D. (2019). *TinyML: Machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*. O'Reilly Media.
9. Banbury, C., Reddi, V. J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., Montino, P., Kanter, D., Ahmed, S., Pau, D., Thakker, U., Torrini, S., Maggioni, M., Aubert, J., & Zhou, A. (2021). MLPerf Tiny benchmark. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (Vol. 1). https://arxiv.org/abs/2106.07597
10. Lin, J., Chen, W.-M., Lin, Y., Gan, C., Han, S., & others. (2021). MCUNetV2: Memory-efficient patch-based inference for Tiny deep learning. In *Advances in Neural Information Processing Systems*. https://arxiv.org/abs/2110.15352
11. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2704–2713). https://doi.org/10.1109/CVPR.2018.00286
12. Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proceedings of the International Conference on Learning Representations*. https://arxiv.org/abs/1510.00149
13. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv*. https://arxiv.org/abs/1503.02531
14. Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2018). Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine, 35*(1), 126–136. https://doi.org/10.1109/MSP.2017.2765695
15. Cai, H., Zhu, L., & Han, S. (2020). Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*. https://arxiv.org/abs/1908.09791
16. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv*. https://arxiv.org/abs/1704.04861
17. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520). https://doi.org/10.1109/CVPR.2018.00474
18. Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 6848–6856). https://doi.org/10.1109/CVPR.2018.00716
19. Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning* (pp. 6105–6114). PMLR. https://proceedings.mlr.press/v97/tan19a.html
20. Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 1314–1324). https://doi.org/10.1109/ICCV.2019.00140