From Tiny Machine Learning to Tiny Deep Learning: A Survey

SHRIYANK SOMVANSHI, Texas State University, San Marcos, United States
MD MONZURUL ISLAM, Texas State University, San Marcos, United States
GAURAB CHHETRI, Texas State University, San Marcos, United States
ROHIT CHAKRABORTY, Texas State University, San Marcos, United States
MAHMUDA SULTANA MIMI, Texas State University, San Marcos, United States
SAWGAT AHMED SHUVO, Texas State University, San Marcos, United States
KAZI SIFATUL ISLAM, Texas State University, San Marcos, United States
SYED JAVED, Texas State University, San Marcos, United States
SHARIF AHMED RAFAT, Texas State University, San Marcos, United States
ANANDI DUTTA, Texas State University, San Marcos, United States
SUBASISH DAS, Civil Engineering, Texas State University, San Marcos, United States

The rapid growth of edge devices has driven the demand for deploying artificial intelligence (AI) at the edge, giving rise to Tiny Machine Learning (TinyML) and its evolving counterpart, Tiny Deep Learning (TinyDL). While TinyML initially focused on enabling simple inference tasks on microcontrollers, the emergence of TinyDL marks a paradigm shift toward deploying deep learning models on severely resource-constrained hardware. This survey presents a comprehensive overview of the transition from TinyML to TinyDL, encompassing architectural innovations, hardware platforms, model optimization techniques, and software toolchains. We analyze state-of-the-art methods in quantization, pruning, and neural architecture search (NAS), and examine hardware trends from MCUs to dedicated neural accelerators. Furthermore, we categorize software deployment frameworks, compilers, and AutoML tools enabling practical on-device learning. Applications across domains such as computer vision, audio recognition, healthcare, and industrial monitoring are reviewed to illustrate the real-world impact of TinyDL. Finally, we identify emerging directions including neuromorphic computing, federated TinyDL, edge-native foundation models, and domain-specific co-design approaches. This survey aims to serve as a foundational resource for researchers and practitioners, offering a holistic view of the ecosystem and laying the groundwork for future advancements in edge AI.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Deep learning**; **TinyML**; *Model interpretability*; • **Applied computing** → *Predictive analytics.*

Additional Key Words and Phrases: Tiny Machine Learning, Tiny Deep Learning, Edge AI, Embedded Deep Learning

Authors' Contact Information: Shriyank Somvanshi, Texas State University, San Marcos, Texas, United States; e-mail: shriyanksomvanshi@gmail.com; Md Monzurul Islam, Texas State University, San Marcos, Texas, United States; e-mail: monzurul@txstate.edu; Gaurab Chhetri, Texas State University, San Marcos, Texas, United States; e-mail: gaurab@txstate.edu; Rohit Chakraborty, Texas State University, San Marcos, Texas, United States; e-mail: rohitchakraborty@txstate.edu; Mahmuda Sultana Mimi, Texas State University, San Marcos, Texas, United States; e-mail: qnb9@txstate.edu; Sawgat Ahmed Shuvo, Texas State University, San Marcos, Texas, United States; e-mail: sawgat@txstate.edu; Kazi Sifatul Islam, Texas State University, San Marcos, Texas, United States; e-mail: kazi_sifat@txstate.edu; Syed Javed, Texas State University, San Marcos, Texas, United States; e-mail: aaqib.ce@txstate.edu; Sharif Ahmed Rafat, Texas State University, San Marcos, Texas, United States; e-mail: sarafat@txstate.edu; Anandi Dutta, Texas State University, San Marcos, Texas, United States; e-mail: anandi.dutta@txstate.edu; Subasish Das, Civil Engineering, Texas State University, San Marcos, Texas, United States; e-mail: subasish@txstate.edu.

ACM 1557-7341/2025/11-ART
https://doi.org/10.1145/3776588

1 Introduction

Tiny Machine Learning (TinyML) has emerged as a rapidly growing paradigm that brings machine learning capabilities to severely resource-constrained edge devices. Traditionally, machine learning models demanded significant computational resources, making their deployment on microcontroller units (MCUs) and embedded platforms impractical. However, advances in hardware design, model compression, and embedded inference have allowed real-time intelligence to be embedded on-device, leading to a new class of systems that execute complex analytics at the edge. As the field evolves, a distinct subdomain called Tiny Deep Learning (TinyDL) has gained momentum, focusing specifically on deploying deep learning models, rather than shallow classifiers on low-power, ultra-constrained hardware.

TinyML is typically defined as the deployment of machine learning inference tasks on devices operating under 1 mW of power, often with only 32 to 512 kB of Static Random-Access Memory (SRAM) and constrained flash storage. These devices, which usually lack an operating system and hardware accelerators for floating-point operations, are capable of performing real-time analytics while meeting stringent energy and memory budgets [1–3]. TinyDL builds upon this foundation by emphasizing the use of deep neural networks, such as convolutional and transformer-based architectures, under similar constraints. This term, introduced as early as 2017 with just-in-time inference frameworks like TinyDL [4], now encompasses a range of state-of-the-art models such as MCUNet, EfficientNet-lite, and DistilBERT variants that deliver strong accuracy with memory footprints below 1 MB and latency below 20 milliseconds [5].

The rise of TinyML and TinyDL is primarily driven by limitations inherent in traditional cloud-based machine learning workflows. Cloud inference introduces unacceptable round-trip latencies in time-sensitive applications such as autonomous driving, drones, and wearables [6]. Moreover, transmitting sensor data to the cloud raises substantial privacy concerns in healthcare and industrial Internet of Things (IoT) contexts, where data sovereignty and user trust are paramount [7]. Finally, the energy consumption required to constantly stream data to remote servers introduces a prohibitive cost, especially for battery-powered devices [8]. By shifting inference-and increasingly, lightweight learning-onto the device, TinyDL enables ultra-low-latency responses, reduces dependency on cloud connectivity, and enhances data privacy [1].

Initially, TinyML systems relied on shallow models such as linear classifiers, decision trees, or single-layer perceptrons. These models, while lightweight, were unable to match the representational power of deep neural networks and required extensive manual feature engineering, particularly for audio and vision tasks [9]. The transition toward TinyDL was made possible by several interrelated advances. First, architectural innovations such as depthwise separable convolutions, inverted residuals, and attention mechanisms made it possible to compress model complexity without sacrificing accuracy [2]. Second, a suite of optimization techniques including quantization-aware training (QAT), structured pruning, knowledge distillation, and low-rank factorization, dramatically reduced the runtime and memory demands of deep models [5]. Third, the introduction of Neural Architecture Search (NAS) frameworks that co-optimize model topology and deployment constraints-such as MCUNet and TinyNAS-has demonstrated that ImageNet-scale tasks can be executed on MCUs with just 480 kB of SRAM [10]. Additionally, new developments in on-device and continual learning allow models to adapt in real-time under strict memory and compute constraints, further extending the practicality of TinyDL systems [11].

1.1 Objectives and Scope of This Survey

This survey aims to provide a comprehensive and timely synthesis of the emerging landscape of TinyDL. While several reviews have previously outlined the evolution of TinyML and its applications up to 2022 [3, 8, 12], they generally focus on classical machine learning or do not sufficiently distinguish TinyDL as a distinct subfield. Our work addresses this gap by emphasizing deep models and techniques tailored for kilobyte-scale environments. We highlight developments occurring through 2025 and offer an integrated perspective that spans model design,

software toolchains, hardware platforms, and deployment strategies. This includes insights from academic research, open-source benchmarks, and industrial deployment case studies. Moreover, we identify critical gaps in current research, such as the lack of support for federated learning, the security of over-the-air updates, and the absence of robust benchmarks for TinyDL systems, and propose a structured agenda for future work.

1.2 Summary of Contributions

To guide this discussion, we contribute a unified definition and taxonomy that clearly delineates TinyDL from traditional TinyML, incorporating hardware constraints and algorithmic characteristics. We offer a comprehensive literature synthesis derived from over 200 sources, structured around recent advances in model architectures, NAS methods, toolchains, and application domains. In addition, we propose a benchmarking framework for evaluating TinyDL systems, incorporating metrics such as inference latency, memory usage, model size, and energy efficiency. As a companion to this paper, we also release `awesome-tinyml`[1], a curated, automatically updated open-source repository of TinyML research papers, tools, frameworks, and tutorials to support community knowledge sharing. Finally, we present a research roadmap that highlights open questions around neuromorphic TinyDL, domain-specific accelerators, compiler–hardware co-design, and privacy-preserving on-device learning. Through these contributions, this survey aims to support both newcomers and experienced researchers in navigating and contributing to the evolving field of TinyDL.

The remainder of this paper is structured as follows: Section 2 introduces TinyML and TinyDL concepts; Section 3 reviews hardware platforms and benchmarks; Section 4 outlines the evolution from TinyML to TinyDL; Section 5 presents lightweight deep learning architectures; Section 6 discusses software toolchains and deployment frameworks; Section 7 highlights key applications across domains; Section 8 explores on-device learning methods; Section 9 covers evaluation metrics and datasets; Section 10 discusses ongoing research challenges; Section 11 suggests future directions; and Section 12 concludes the paper.

2 Background and Foundational Concepts

2.1 Tiny Machine Learning

TinyML has emerged as a transformative field within artificial intelligence, characterized by the deployment and execution of machine learning models on highly resource-constrained embedded devices, particularly MCUs. This paradigm facilitates on-device data processing and inference, thereby pushing intelligence to the very edge of networks [8, 12]. The fundamental aim of TinyML is to enable sophisticated analytical capabilities directly on hardware platforms that are severely limited in terms of their available resources. In doing so, it supports applications requiring low latency, minimal power consumption, and enhanced data privacy by keeping data local [8, 9, 13].

The operational landscape of TinyML is shaped by three critical constraints: memory, power, and compute limitations. Firstly, memory availability is exceptionally scarce. Devices typically include SRAM ranging from a few kilobytes to several hundred kilobytes for runtime operations, and Flash memory often under one megabyte for storing program code and ML models, though in rare cases this may reach up to 2 MB [9, 12, 14–16]. This represents a stark contrast to conventional computing platforms and necessitates the use of highly compact models [14]. Secondly, power consumption is a paramount constraint. TinyML devices typically operate on ultra-low power budgets, often in the milliwatt or even microwatt range [8, 9, 12, 15, 16]. This is essential for battery-powered or energy-harvesting systems designed for long-term operation without frequent recharging or maintenance [8, 15]. Thirdly, compute capabilities in these devices are limited. The MCUs generally operate at clock speeds of several tens to a few hundred megahertz, and many lack Floating Point Units (FPUs), which further constrains the deployment of typical ML models unless optimized through quantization techniques

[1] https://github.com/gauravfs-14/awesome-tinyml

[9, 12, 16]. These hardware limitations necessitate lightweight and efficient models capable of running within constrained environments.

The hardware ecosystem supporting TinyML primarily consists of low-power MCUs integrated with sensors that gather environmental data. Prominent examples include the ARM Cortex-M series, such as the Cortex-M0, M4, and M7, which strike a balance between computational efficiency, power consumption, and cost [3, 9, 17]. Other widely used platforms include the STM32 and ESP32 families [8, 16]. These MCUs are often paired with application-specific sensors, such as inertial measurement units (IMUs) for motion tracking, microphones for voice command recognition, and low-resolution cameras for vision tasks with constrained compute budgets [3, 9]. This combination of efficient hardware and targeted sensors empowers TinyML to bring intelligence into everyday objects, from wearables to smart infrastructure.

2.2 Tiny Deep Learning

TinyDL represents a specialized subfield within the broader TinyML domain, specifically concentrating on the adaptation and deployment of deep learning models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), onto extremely resource constrained hardware, most notably MCUs [5, 18]. The primary objective of TinyDL is to enable sophisticated tasks like image classification, object detection, and real time gesture recognition on these low power devices [16, 19].

A key characteristic of TinyDL is the utilization of highly compressed deep models. These models are meticulously optimized to fit within the stringent memory limitations of MCUs, often resulting in model sizes of just a few hundred kilobytes [20, 21]. This substantial reduction is achieved through various model compression techniques. Quantization, for instance, involves converting the model's parameters from higher precision floating point numbers to lower precision integers, such as 8 bit integers, thereby shrinking the model size with minimal degradation in accuracy [14, 16, 19]. Another prevalent technique is pruning, which systematically removes redundant parameters or connections within the neural network to create sparser and more compact models [5, 16]. Furthermore, TinyDL models are designed for real time inference. This means they can process data and provide outputs almost instantaneously on the device itself, which is crucial for applications requiring immediate responses [16, 19].

TinyDL differs significantly from conventional Deep Learning. Conventional DL typically relies on powerful computing resources like GPUs and extensive memory (often gigabytes) to train and run large, complex models, with the primary goal of achieving the highest possible accuracy [16, 17]. In contrast, TinyDL operates under severe hardware limitations, prioritizing on device efficiency, low power consumption (milliwatts or microwatts), and minimal memory footprint (kilobytes) [5, 16]. Recent advancements in TinyDL have introduced neural network architectures specifically designed for edge execution, including MobileNet, SqueezeNet, and Tiny-YOLO [22–24]. These models are tailored to execute with fewer floating-point operations and reduced parameter counts, enabling inference in real time even on MCUs without FPUs [12, 16]. Furthermore, hardware-aware NAS and energy-efficient training paradigms are being actively explored to enhance model deployment on edge platforms [8]. Within the context of TinyML, TinyDL is a specialized area. TinyML encompasses all machine learning techniques, including classical algorithms, that can be deployed on resource-limited devices [25]. TinyDL, however, specifically addresses the more demanding challenge of implementing and running inherently more complex and resource intensive deep learning models under these same severe constraints [5, 20].

2.3 Workflow in TinyML and TinyDL

The development of TinyML solutions involves distinct workflows that bridge the creation of machine learning models with their deployment on resource constrained hardware. Three primary approaches are recognized in the literature: the ML oriented workflow, the hardware oriented workflow, and the co design workflow [12].These

workflows are illustrated in Figure 1, which provides a comparative overview of the design focus, optimization stages, and implementation flow for each approach. The ML oriented workflow is primarily driven by machine learning practitioners. This approach commences with the design, training, and validation of an ML model suited for the specific problem, often initially disregarding the precise limitations of the target hardware to maximize performance and generalization [12, 15]. Following this, the model undergoes an optimization phase, where techniques like pruning and quantization are applied to reduce its size and computational demands to meet the hardware constraints [12, 15]. The final steps involve deploying the optimized model to the target device and evaluating its real world performance [12].

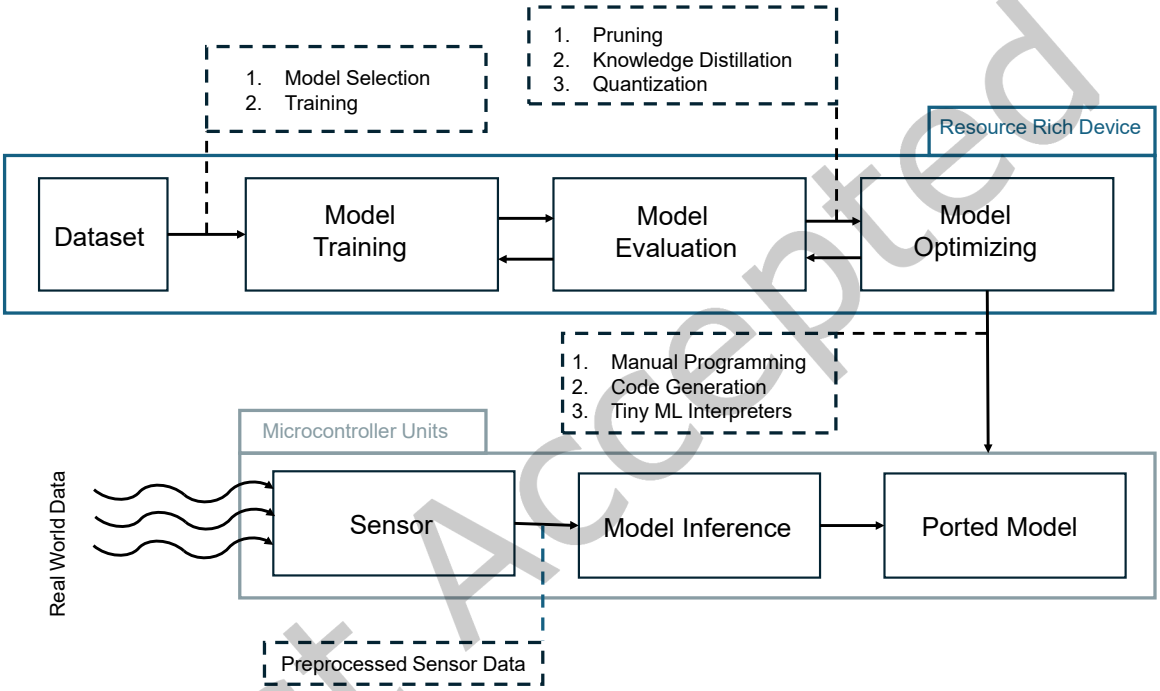

Fig. 1. TinyML Pipeline [15]

Conversely, the hardware oriented workflow is led by hardware engineers. The main focus here is on designing new, or enhancing existing, hardware platforms such as MCUs or specialized accelerators, to execute ML algorithms with greater efficiency [12]. This workflow involves identifying computational bottlenecks in current hardware when running ML tasks and then architecting specific hardware solutions to mitigate these issues, thereby improving throughput and reducing power consumption for ML workloads [12]. The co design workflow represents a more integrated and increasingly pivotal approach. In this paradigm, ML experts and hardware engineers collaborate closely from the project's inception [12]. Unlike the sequential nature of the ML oriented or HW oriented workflows, co design involves the intertwined and simultaneous optimization of both the ML model and the hardware architecture [5, 12]. This holistic methodology aims to achieve optimal synergy between software and hardware, potentially unlocking performance and efficiency gains unattainable through isolated optimization efforts, and is considered crucial for advancing the capabilities of TinyML systems [12]. Furthermore,

a comparative summary of TinyML, TinyDL, and Edge AI paradigms is presented in Table 1, highlighting their design scope, hardware requirements, model complexity, and implementation focus.

Table 1. Comparison of TinyML, TinyDL, and Edge AI

Feature	TinyML	TinyDL	Edge AI
Scope	Subset of Edge AI; ML on extremely resource-constrained devices, typically MCUs [9, 12]	Subset of TinyML; specifically deploying deep learning models on these extremely resource-constrained MCUs [5, 20]	Broadest category; AI processing near the data source, on devices ranging from gateways and edge servers to MCUs [8, 26]
Typical Hardware	MCUs (e.g., ARM Cortex-M series, ESP32), DSPs [9, 16]	Same MCUs, occasionally with minimal accelerators [17, 19]	Edge servers, GPUs (e.g., NVIDIA Jetson), FPGAs, powerful SoCs, capable MCUs/MPUs [8, 9]
Model Complexity	Classical ML and highly compressed DL models (kB to $\lesssim$few MB) [9, 14]	Deep networks heavily quantized or pruned to fit kB-scale memory [19, 20]	Ranges from simple ML to complex DL, hardware-dependent [8]
Primary Goals	Enable ML on ultra-low-power, low-cost devices; maximize battery life; ensure data privacy [8, 9]	Bring sophisticated DL to the most constrained devices, pushing efficiency limits [5, 16]	Reduce latency and bandwidth, improve privacy, enable real-time analytics at the edge [8, 26]
Power Consumption	Typically in the mW–µW range [8, 16]	Same as TinyML, with tight per-inference budgets [5, 17]	Wide range: watts (edge servers) to milliwatts (embedded) [8]
Data Processing	Strictly on-device at MCU level [9]	Same scope, on-device MCU inference [19]	Local edge servers, gateways, or capable end devices [8]
Key Characteristic	ML at the "tiniest" compute scale, adding intelligence to everyday objects [9]	Neural networks at kB-scale, requiring extreme optimization [19]	Decentralized AI that moves computation out of the cloud [26]
Relationship	Specialized subset of Edge AI, focused on the resource-limited extreme [8]	Specialized subset of TinyML, centered on deep learning algorithms [20]	Superset covering non-cloud AI workloads [26]

Figure 2 summarizes the transition from TinyML to TinyDL across four key dimensions: model size, hardware platforms, optimization techniques, and application domains. TinyML typically employs classical machine learning models under 250 KB on low-power MCUs, whereas TinyDL enables the deployment of compressed deep learning models through techniques such as quantization-aware training (QAT), neural architecture search (NAS), hardware-aware quantization (HAQ), and knowledge distillation. This evolution expands practical use cases from simple tasks like gesture and ECG monitoring to more complex applications such as speech recognition, computer vision, and autonomous systems.

3 Hardware Ecosystem

TinyML and TinyDL applications run on highly resource-constrained devices. This section surveys the hardware platforms enabling TinyML, from general MCUs to new specialized AI accelerators, and how these platforms are evaluated. Despite their modest specs, these devices can perform meaningful ML tasks at the edge by balancing performance, power, and accuracy through careful design and benchmarking.

3.1 MCUs

MCUs form the backbone of many TinyML deployments, bringing intelligence to the extreme edge. They are single-chip computers designed for low power operation, often running on batteries in remote sensors or wearables [27]. MCUs operate under strict hardware constraints: low clock speeds (typically on the order of tens of MHz) and limited on-chip memory (often only tens to a few hundred kilobytes of RAM). For example, a typical MCU might have approximately 128 KB of RAM and 1 MB of flash storage, versus the gigabytes of memory and storage on a modern smartphone [28]. Because they usually run on small batteries, energy efficiency is

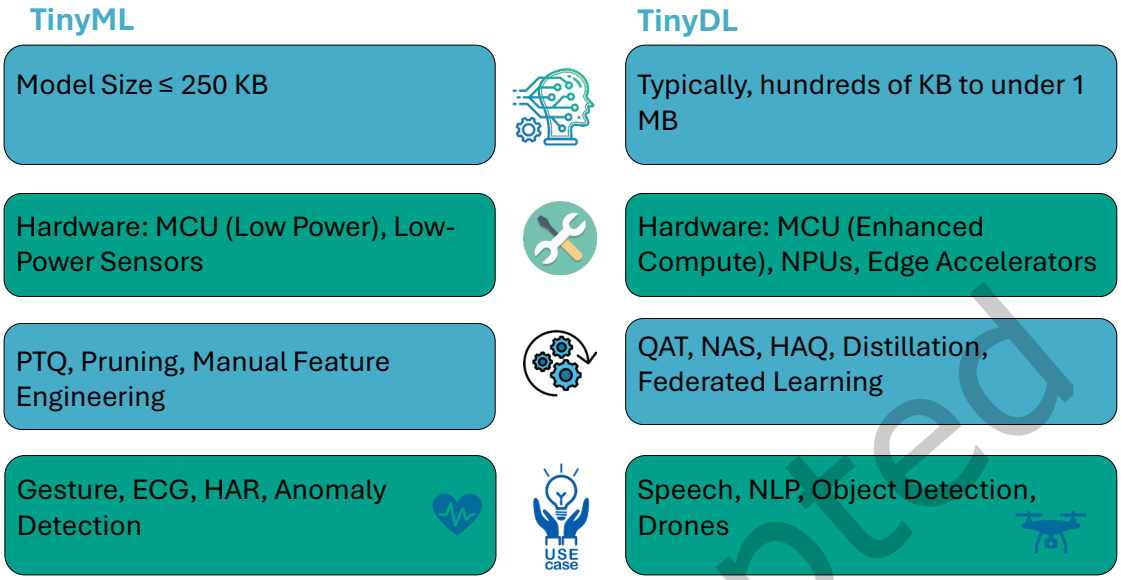

Fig. 2. Comparative summary of TinyML and TinyDL across four key aspects: model size, hardware platforms, optimization techniques, and representative use cases

paramount – TinyML devices must consume mere milliwatts or microwatts to run for long periods [27]. Despite these limitations, MCUs are capable of running surprisingly complex ML workloads when models are optimized. Continuous improvements in MCU hardware (e.g. more efficient 32-bit ARM Cortex-M processors) have made them powerful and energy-efficient enough to handle small neural networks within tight power budgets [17]. In other words, it is now feasible to deploy machine learning on battery-operated MCU-based sensors in the field. Equally important is the software toolchain that supports MCUs. Frameworks like TensorFlow Lite for MCUs (TFLite Micro) and platforms like Edge Impulse allow developers to compress and deploy trained models on tiny devices. For instance, TFLite Micro can convert a neural network into a form that runs in as little as 16 KB of RAM on an MCU. Techniques such as 8-bit quantization and pruning are used to shrink model size and computation so that inference can execute in real time on limited CPU and memory.

Key Hardware Constraints for MCU-based TinyML

Clock Speed: Often 20-200 MHz clock rates (much lower than GHz-class processors) [27], which limits the raw compute throughput of MCUs.
Memory: On the order of kilobytes to a few hundred kilobytes of RAM and usually under a few MB of flash storage [29]. This means models must be very small and efficient.
Power/Battery: Designed for ultra-low power use; many MCU systems run on coin-cell batteries or energy harvesters. Power consumption is in the milliwatt or even microwatt range, so the device can operate for months or years [29].

Even with these constraints, MCUs have demonstrated the ability to run useful ML inference tasks at the edge. By using optimized models, an MCU can perform tasks like keyword spotting (KWS), gesture recognition, anomaly detection, or simple image classification entirely on-device [30]. For example, researchers have successfully deployed a voice wake-word detector and simple vision models on tiny boards like the Espressif ESP32 and STMicroelectronics STM32 series MCUs. The ESP32 (a dual-core MCU up to 240 MHz with approximately 520 KB RAM) and various STM32 Cortex-M variants (e.g. an M7 at 216 MHz with a few hundred KB of RAM) are popular choices that, with quantized models, can handle basic deep learning tasks under tight memory and energy constraints. Recent studies and white papers highlight that the combination of improving MCU hardware and clever model optimizations enables these chips to support machine learning workloads that were once thought impossible on such limited devices [27]. MCUs provide a flexible, low-cost platform for TinyML, albeit one that demands extreme efficiency in model design.

3.2 Specialized AI Hardware

While many TinyML applications run on general-purpose MCUs, a new class of specialized AI hardware has emerged to push the boundaries of performance and efficiency for tiny and edge deployments. These are application-specific chips and co-processors built specifically to handle neural network inference with minimal energy. By using custom digital logic (and in some cases analog techniques), they act as Neural Compute Engines that drastically accelerate ML tasks compared to a software-only MCU approach. The following subsubsections describe notable examples of AI-focused hardware designed for low-power, on-device deep learning:

3.2.1 Google Edge Tensor Processing Unit. The Google Edge Tensor Processing Unit (TPU) is a small application-specific integrated circuit (ASIC) designed by Google to accelerate TensorFlow Lite models at the edge. Each Edge TPU can perform 4 trillion operations per second (4 TOPS) while consuming about 2W of power (roughly 2 TOPS/W) [31]. In practical terms, an Edge TPU can run vision models like MobileNet V2 at nearly 400 frames per second in a power-efficient manner [32], far beyond what a typical MCU could achieve. These chips often come as co-processors (e.g., in the Coral EdgeTPU USB sticks or M.2 modules) that pair with a MCU or microprocessor, offloading the heavy math of neural networks. By handling matrix multiplications and convolutions in dedicated hardware, the EdgeTPU enables real-time image and audio inference on the edge device with minimal latency and modest power use.

3.2.2 Syntiant Neural Decision Processors Series. Syntiant's Neural Decision Processors (NDP) are ultra-low-power neural accelerators aimed at always-on workloads like KWS and sensor analytics. They use a custom deep neural network inference engine that runs models efficiently with parallel multiply-accumulate (MAC) units and an optimized data path for minimal idle cycles [31]. For example, the Syntiant NDP120 can continuously listen for voice commands using only a few microwatts. In MLPerf Tiny benchmark tests, the NDP120 was able to perform a KWS inference in about 4.3 ms while consuming only 35 μJ of energy per inference (at 30 MHz operation) [31]. This is orders of magnitude more energy-efficient than running the same task on a generic MCU. The NDP chips achieve this by being ASICs optimized for neural workloads-they store and process neural network layers on-chip to avoid costly memory accesses, and they integrate small digital signal processor (DSP) cores for preprocessing tasks. Syntiant's platform demonstrates how specialized silicon can deliver real-time AI within a milliwatt power budget.

3.2.3 Himax WiseEye WE-I Plus. The Himax WE-I Plus (HX6537-A) is an example of an AI-enabled MCU/ASIC tailored for vision and sensor inferencing at the edge. It combines a 400 MHz DSP with dedicated hardware accelerators (for tasks like image processing, HOG feature extraction, and JPEG encoding) in an ultra-low-power design [33]. Uniquely, the WE-I Plus is event-driven: it stays in a near-standby mode until its camera or sensor accelerator detects a trigger (e.g., motion or a person in view), then the DSP wakes to run a neural network

inference [33]. This architecture is highly power-efficient. In fact, when running a person-detection CNN (TinyML vision model), the average power consumption can be under 5 mW-an exceptionally low figure for an image recognition task. By leveraging an ASIC with built-in neural accelerators, the Himax WE-I Plus achieves real-time vision inference (e.g., detecting human presence in a frame) using only a fraction of the energy that a general-purpose MCU would require for the same task [33].

These examples illustrate the importance of custom AI silicon for TinyML. Specialized edge AI chips like the Edge TPU, Syntiant NDP, Himax WE-I, as well as others (e.g. Intel's Movidius Myriad X visual processing unit and various analog neural chips), focus on the common computational patterns of ML algorithms. By implementing neural network operations (matrix multiplies, convolutions, etc.) in hardware, they achieve far higher throughput per watt than a CPU. Innovations such as parallel MAC arrays, on-chip memory for weights/activations, and streamlined dataflows allow these ASICs to perform inference with minimal wasted energy. Many also include features like built-in DSPs or camera interfaces to handle sensor data directly. The result is low-power, real-time inference: tasks like wake-word detection or gesture recognition can run continuously on the edge without exhausting a battery. Table 2 summarizes key TinyML hardware platforms, including MCUs and neural accelerators, along with their specifications and typical use cases.

3.3 Benchmarking and Evaluation

Standardized benchmarking frameworks play a critical role in enabling fair and consistent evaluation of TinyML hardware platforms. They provide a unified basis for comparing different systems under constraints of speed, power consumption, and inference accuracy. Two widely adopted benchmark suites in this space are MLPerf Tiny and EEMBC MLMark.

MLPerf Tiny, developed by MLCommons in collaboration with EEMBC, is specifically designed for ultra-low-power AI systems. It defines four core tasks representative of common TinyML applications: keyword spotting (KWS), Visual Wake Words (VWW), image classification on low-resolution inputs, and anomaly detection using sensor data [34]. These tasks are performed using compact neural networks, typically under 250 kB in size. MLPerf Tiny evaluates system performance in a single-stream inference mode, mimicking real-time sensor workloads. It reports latency per inference and model accuracy, and also includes an optional energy consumption metric through EEMBC's EnergyRunner harness [34]. This allows the benchmark to quantify both throughput and power efficiency (e.g., energy per inference), enabling comprehensive comparisons across platforms.

EEMBC MLMark serves a broader purpose by benchmarking general embedded ML inference. It provides a standardized methodology for measuring inference latency and accuracy using fixed models and datasets, ensuring reproducibility across different hardware platforms [35]. All implementations must use provided test harnesses or disclose optimizations, enforcing a level playing field. While MLMark does not include built-in energy metrics—delegating such measurements to benchmarks like EEMBC's ULPMark for microcontroller efficiency—it remains valuable for assessing system throughput and model performance [36].

Together, MLPerf Tiny and MLMark offer complementary strengths. MLPerf Tiny incorporates energy-aware metrics essential for power-constrained TinyML deployments, while MLMark provides broader coverage and a reproducible baseline for embedded ML systems. For example, MLPerf Tiny may reveal that a specialized neural accelerator achieves five times lower latency or ten times greater energy efficiency than a baseline MCU when running the same KWS model [31]. Similarly, MLMark can track accuracy and inference improvements as new microcontrollers, neural processing units (NPUs), or software frameworks are introduced [35].

These benchmarking tools have become indispensable for researchers and system designers. They enable rigorous, quantitative evaluation of performance, energy usage, and inference quality. As TinyML continues to evolve, these frameworks are expected to adapt by supporting larger models, more diverse sensor modalities,

and increasingly fine-grained energy profiling. Such standardized evaluation ensures that TinyML hardware solutions remain responsive to real-world application demands in resource-constrained environments.

Table 2. Examples of TinyML Hardware Platforms and Their Characteristics

Platform	Type	Processor / Clock	Memory (RAM/Flash)	Notable Features	Typical TinyML Use Case
Espressif ESP32	MCU (Wi-Fi SoC)	Dual-core 32-bit MCU @ 240 MHz [27]	520 KB SRAM, 4 MB flash (external)	Wi-Fi/Bluetooth integrated; low cost	IoT sensors, simple KWS
STM32 (e.g., STM32F7)	MCU (Cortex-M7)	216 MHz ARM Cortex-M7 MCU [27]	~320 KB RAM, 1 MB flash	DSP instructions, optional FPU	Industrial sensing, audio classification
Google EdgeTPU	ASIC Neural Accelerator	Custom ASIC @ ~200 MHz (equiv.)	Uses host memory (external DRAM)	4 TOPS (~400 FPS MobileNet) at 2 W PCIe/USB interfaces [32]	High-speed vision (object detection, etc.)
Syntiant NDP120	Neural co-processor ASIC	Programmable DNN core @ 30–100 MHz	On-chip memory for models	~35 μJ per inference (KWS); always-on capability [31]	Always-listening AI (wake word, anomaly detection)
Himax WE-I Plus	AI MCU/ASIC with DSP	400 MHz DSP + accelerators [33]	2 MB SRAM, 4 MB flash (typical dev board)	Camera interface; <5 mW person detection [33]	Ultra-low-power vision (people counting, etc.)

4 Evolution from TinyML to TinyDL

4.1 Limitations of Classical TinyML

4.1.1 Poor Generalization for Vision/Audio. Generalization error refers to the difference in performance between a model's training and test datasets. Although this metric is commonly used, it may not fully capture real-world performance, especially when training and test sets come from similar distributions (e.g., same user or device). Research shows that large deep neural networks, despite their capacity to memorize data, often exhibit low generalization error [37]. In contrast, TinyML models, due to their limited capacity and computational constraints, typically exhibit higher generalization errors. Their reduced ability to learn complex features makes them more prone to poor performance on unseen or out-of-distribution data. For instance, in the VWW task, lightweight convolutional models such as MobileNet and MCUNet have achieved 85–90% accuracy within 200–250 KB memory budgets [38, 39]. In contrast, traditional pipelines using handcrafted features like HOG with classical classifiers (e.g., SVM, decision trees) tend to perform significantly worse-typically in the 70–75% accuracy range-due to their limited ability to capture spatial hierarchies and generalize to real-world images under constrained memory [31, 40].

4.1.2 High Feature Engineering Burden. Feature engineering involves manually selecting or transforming raw sensor data (e.g., audio, motion, temperature) into meaningful inputs for traditional models like decision trees or SVMs. While essential, this process is time-consuming, requires domain expertise, and often fails to capture complex patterns as effectively as deep learning. In TinyML, these limitations are amplified due to the resource constraints of edge devices, making manual pipelines impractical for real-time, scalable applications.

4.2 What has Enabled TinyDL

4.2.1 Compression Breakthroughs. Model compression techniques aim to achieve a more efficient representation of one or more layers in a neural network, often with a potential trade-off in quality [41]. These techniques

reduce the model's size and computational requirements, leading to a 20% to 30% decrease in memory usage [8]. There are several general model compression strategies, such as pruning, low-rank factorization, and knowledge distillation. In addition to these, Ray [9] analyzed specific model compression techniques, including the Tiny Anomaly Compressor [42], Doped Kronecker Product [43], and Starfish [44], particularly in the context of image compression.

Tiny Anomaly Compressor offers a lightweight, model-agnostic compression method that is well-suited for on-device anomaly detection in MCU-based IoT systems, though it faces challenges related to validity and generalizability. Doped Kronecker Product enhances traditional Kronecker Product compression by mitigating accuracy loss in Natural Language Processing (NLP) tasks through co-matrix regularization. Meanwhile, Starfish presents a loss-resilient, AutoML-optimized framework designed for efficient image compression and streaming in resource-constrained IoT environments.

4.2.2 Quantization and Hardware Advances. Quantization is a cornerstone of TinyDL, enabling significant model compression and computational efficiency by representing weights and activations in reduced-precision formats (typically INT8 or lower). This approach reduces memory footprint and multiply–accumulate (MAC) operations, making deployment feasible on resource-constrained MCUs. Figure 3 illustrates the TinyDL hardware ecosystem, highlighting core compute units (MCUs, DSPs, NPUs), memory elements (SRAM, Flash, DRAM), and interface components.

Modern quantization techniques go beyond simple format conversion. Hardware-aware approaches and novel numeric representations are increasingly used to optimize for latency and energy. The Cortex-M4, with its Single Instruction Multiple Data/DSP support, remains the most widely adopted MCU in TinyML due to its balance of efficiency and ecosystem maturity. For performance-critical tasks, NPUs accelerate matrix operations with far greater energy efficiency than general-purpose processors.

Key Quantization Approaches Enabling TinyDL

Post-Training Quantization (PTQ): One-shot float → INT8 conversion; ∼4× size drop with negligible accuracy loss [9].
Quantization-Aware Training (QAT): Simulates quant noise during training, enabling 4- or even 2-bit inference while preserving accuracy [45].
Mixed-Precision Schemes (e.g., hardware-aware quantization (HAQ)): Per-layer bit-width selection (2/4/8 bit) based on latency/energy targets [46].
Custom Numeric Formats (e.g., TENT): Tapered or block-floating formats tuned per layer; up to 31% energy savings over INT8 baselines [47].

4.2.3 Lightweight DL Architectures. Deploying deep neural networks on resource-constrained IoT devices poses significant challenges, particularly in NAS. NAS typically involves three key steps: (i) defining the search space of possible architectures, (ii) applying a search algorithm to find the optimal model, and (iii) using an evaluator to balance accuracy and efficiency for deployment [9].

Recent approaches like evolutionary algorithms, differentiable architecture search, progressive search, and parameter sharing have significantly reduced NAS computation costs-from thousands to just a few GPU days-while enabling multi-objective optimization that balances accuracy with efficiency. Techniques such as MNasNet, FBNet, and MONAS further advance this by incorporating latency, power, and computational constraints into the search process, resulting in highly efficient models tailored for deployment on resource-constrained devices [41].

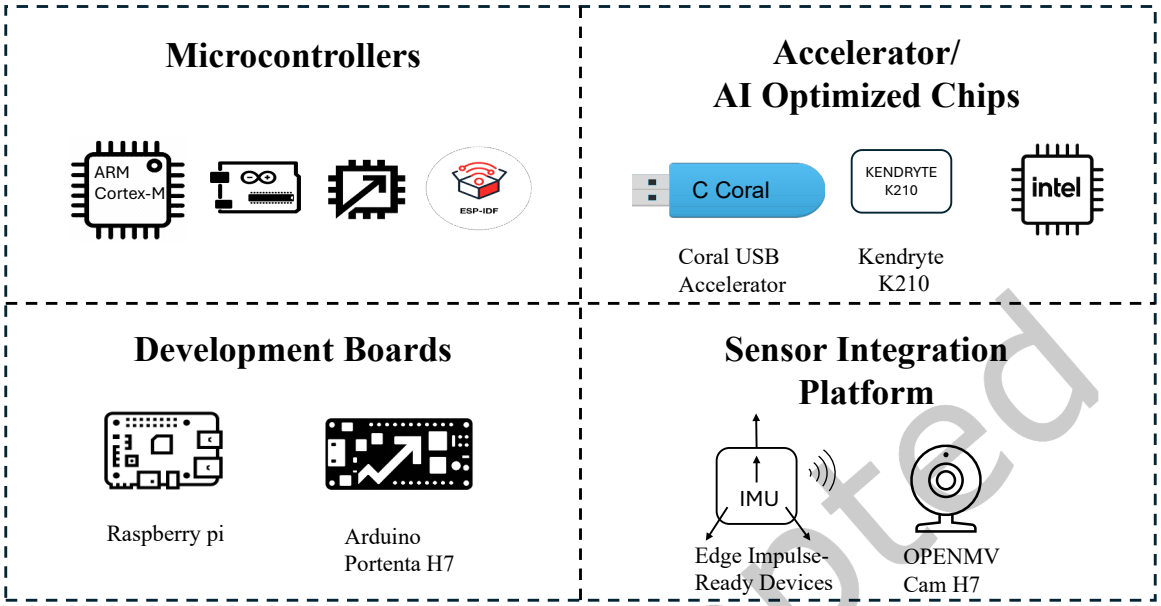

Fig. 3. Hardware Ecosystem of TinyDL

4.3 Key Milestones in TinyDL

4.3.1 MobileNet, TinyBERT, MCUNet, SqueezeNet, DistilBERT. MobileNet, SqueezeNet, and MCUNet are lightweight models for image tasks, designed to run on devices with limited memory. MobileNet uses depthwise separable convolutions, while SqueezeNet uses fire modules to reduce size. MCUNet goes further by running deep learning on tiny MCUs. For language tasks, TinyBERT and DistilBERT are smaller, faster versions of BERT. TinyBERT uses teacher-student training to keep accuracy high, and DistilBERT keeps 97% of BERT's performance with fewer parameters. Figure 4 summarizes key TinyDL breakthroughs from 2016 to 2024, illustrating the progression from early lightweight CNNs like SqueezeNet and MobileNet to advanced models such as MCUNet, TinyBERT, and RedMule that enable deep learning on microcontrollers.

5 TinyDL Architectures and Techniques

The evolution from traditional machine learning to deep learning on resource-constrained devices represents a fundamental paradigm shift in edge computing [56]. This section examines the architectural innovations and optimization techniques that have enabled the deployment of sophisticated deep learning models on MCU-class devices with severe memory and computational limitations.

5.1 Lightweight CNNs

The development of lightweight CNNs specifically designed for resource-constrained environments has been instrumental in enabling deep learning capabilities on edge devices [57]. These architectures employ novel design principles that dramatically reduce computational complexity while maintaining competitive accuracy levels.

5.1.1 MobileNet Architecture Family. The MobileNet series represents a cornerstone achievement in efficient CNN design, introducing depthwise separable convolutions that factorize standard convolutions into depthwise

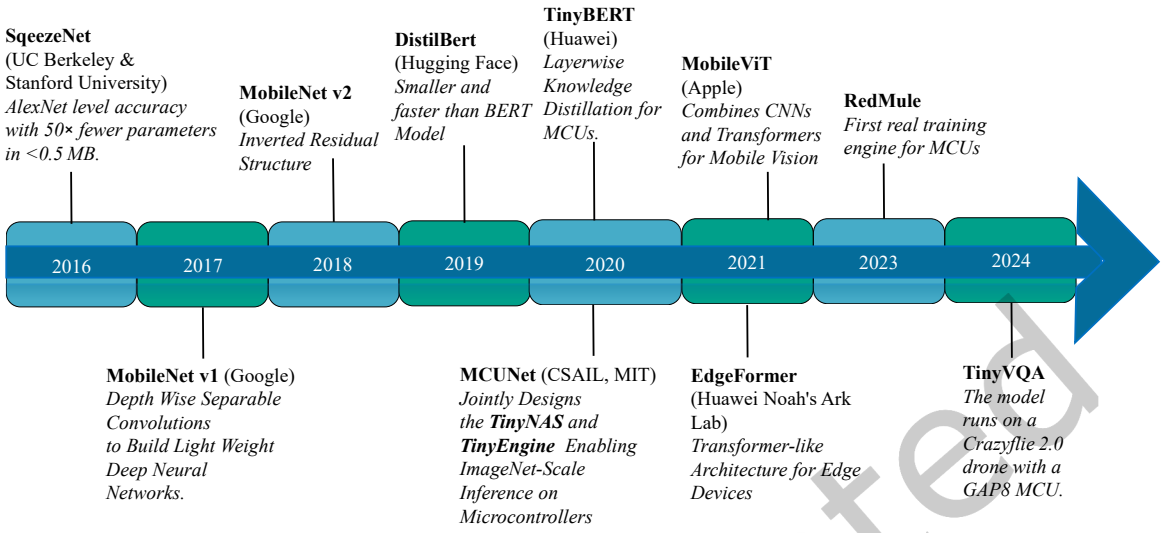

Fig. 4. Timeline of Major TinyDL Breakthroughs [48],[49],[50],[51],[52],[39],[53],[50],[54],[55]

and pointwise operations [57]. MobileNetV2 extends this approach with inverted residual blocks and linear bottlenecks, achieving 71.8% ImageNet top-1 accuracy with only 3.4 MB model size when deployed on STM32H7 MCUs [57]. The architecture's efficiency stems from its fundamental redesign of the convolution operation, reducing parameters by an order of magnitude compared to traditional CNNs while maintaining representational capacity [57].

5.1.2 SqueezeNet and Fire Modules. The SqueezeNet architecture employs fire modules consisting of squeeze layers (1×1 convolutions) followed by expand layers (mixed 1×1 and 3×3 convolutions) to achieve significant parameter reduction [58]. SqueezeNext further optimizes this design with hardware-aware modifications, achieving AlexNet-level accuracy with 112× fewer parameters [58]. The architecture's aggressive parameter reduction makes it particularly suitable for flash-constrained MCUs, requiring only 1.2 MB storage while maintaining 57.5% ImageNet accuracy [59].

5.1.3 Hardware-Aware Architecture Design. Recent developments in lightweight CNN design have emphasized hardware-aware optimization through NAS specifically tailored for MCU constraints [60]. MCUNet demonstrates this approach by achieving 68.7% ImageNet accuracy with only 0.51 MB model size through joint optimization of network architecture and inference scheduling [39]. The framework employs a two-stage NAS that first optimizes the search space to fit resource constraints, then specializes the network architecture within the optimized space [39].

5.2 Lightweight Transformers and RNNs

The adaptation of transformer architectures for TinyML represents a significant advancement in bringing state-of-the-art natural language processing capabilities to edge devices, though it presents unique challenges due to the attention mechanism's quadratic memory complexity [52].

5.2.1 TinyBERT Knowledge Distillation. Representing a breakthrough in transformer compression, TinyBERT employs a novel two-stage knowledge distillation framework specifically designed for transformer models [52].

The approach performs transformer distillation at both pre-training and task-specific learning stages, enabling effective knowledge transfer from large teacher models to compact student networks [52]. TinyBERT-4L achieves 96.7% performance of BERT-Base on the GLUE benchmark while being 7.5× smaller and 9.4× faster on inference, with only 14.5 million parameters [52].

5.2.2 DistilBERT Compression Strategy. Utilizing a different distillation approach, DistilBERT reduces the original BERT model by 40% while retaining 97% of its language understanding capabilities [52]. The model achieves 91.3% F1 score on SQuAD v1.1 with 66 million parameters, demonstrating the effectiveness of student-teacher training with temperature-scaled softmax distributions [52].

5.2.3 MCU Deployment Challenges. Recent research has focused on optimizing transformer deployment specifically for MCU units, addressing unique challenges posed by the multi-head self-attention mechanism [61]. The primary bottlenecks include high memory footprint of intermediate attention results and frequent data marshaling operations [61]. Novel approaches such as Fused-Weight Self-Attention (FWSA) and Depth-First Tiling have been developed to mitigate these challenges, achieving up to 6.19× reduction in memory peak usage while maintaining computational accuracy [61].

A comparative summary of popular TinyDL models, including their architectural characteristics, deployment efficiency, and hardware targets, is provided in Table 3. This table highlights the trade-offs between accuracy, model size, and latency across a diverse range of architectures and deployment contexts, offering valuable insights for selecting appropriate models in resource-constrained scenarios.

Table 3. Summary of TinyDL Models with Size, Inference Speed, and Task Accuracy

Model Name	Architecture	Size (MB)	Latency (ms)	Accuracy (%)	Target Task	Hardware
TinyBERT-4L [52]	Transformer	14.5	5.0	96.8 (SST-2)	Text Classification	Mobile SoC
DistilBERT [52]	Transformer	66.0	7.0	91.3 (SQuAD)	QA / NLP Tasks	Mobile GPU
MobileNetV2-0.35 [57]	CNN	3.4	~32	71.8 (ImageNet)	Image Classification	STM32H7 MCU
SqueezeNet v1.1 [58, 59]	CNN	4.8	~20	58.38 (ImageNet)	Object Detection	Kendryte K210
MCUNet-256kB [39]	CNN + NAS	0.51	12.0	70.7 (ImageNet)	Image Classification	STM32F746
EfficientNet-Lite0 [57]	CNN	4.7	45.0	75.1 (ImageNet)	Image Classification	EdgeTPU
DS-CNN (MLPerf) [31]	1D-CNN	0.05	20.0	>90.0 (Commands)	Wake Word Detection	ARM Cortex-M4
MobileNet (VWW) [31]	CNN	0.32	8.0	80.0 (VWW)	Visual Wake Words	STM32 MCU
Deep AutoEncoder (AD) [31]	Autoencoder	0.27	15.0	85.0 (AD Bench)	Anomaly Detection	MCU Platform
ResNet (IC) [31]	CNN	0.096	25.0	85.0 (ImageNet)	Image Classification	STM32 MCU
Transformer-FWSA [61]	Transformer	2.1	180.0	78.2 (NLP Tasks)	NLP Tasks	STM32F746
SquishedNet [58]	CNN	0.95	156.0	77.0 (CIFAR-10)	Image Classification	Nvidia Jetson TX1

5.3 Model Optimization Techniques

The deployment of deep learning models on TinyML systems necessitates sophisticated optimization techniques that compress model size and accelerate inference while preserving accuracy [62].

5.3.1 Quantization Methodologies. Quantization represents one of the most effective approaches for model compression in TinyML systems [63][64]. PTQ converts trained models from floating-point to reduced precision representations, typically INT8, achieving 4× model size reduction with minimal accuracy degradation [62]. QAT incorporates quantization effects during training, enabling more aggressive precision reduction while maintaining model performance [63]. Comparative analysis shows that quantization generally outperforms pruning across various compression ratios, with benefits becoming more pronounced at moderate compression levels [64].

5.3.2 Neural Network Pruning. Neural network pruning eliminates redundant or less important parameters to reduce model complexity and memory footprint [65][62]. Magnitude-based pruning removes weights with smallest absolute values, providing a straightforward approach for parameter reduction [65]. Structured pruning targets entire network components such as filters or layers, enabling more significant architectural simplifications suitable for severely resource-constrained environments [62]. Research demonstrates that structured pruning can achieve compression ratios up to 13× without significant accuracy loss through iterative pruning and retraining cycles [65].

5.3.3 Joint Optimization Approaches. The combination of quantization and pruning techniques has emerged as a powerful strategy for achieving maximum compression efficiency [63]. Quantization-aware pruning yields more computationally efficient models than either technique alone, particularly for ultra-low latency applications [63]. Joint optimization frameworks demonstrate superior computational efficiency compared to sequential application of compression techniques, with benefits varying based on target compression ratios and application requirements .

5.3.4 Network Augmentation and Auxiliary Supervision. Network augmentation offers a complementary training-time optimization strategy, particularly relevant for TinyDL. As shown in Figure 5, a tiny model is embedded into larger networks that share weights and provide auxiliary supervision. This enables the tiny model to learn stronger representations without increasing its inference-time footprint, making it ideal for resource-constrained deployments.

6 Software Toolchains and Deployment Frameworks

The deployment of TinyML and TinyDL models on resource-constrained devices such as MCUs and edge processors requires robust, efficient, and highly optimized software toolchains. These toolchains bridge the gap between trained machine learning models and their real-world deployment on ultra-low-power hardware. This section examines lightweight deployment frameworks, compilation techniques, and end-to-end platforms that enable practical and scalable TinyML and TinyDL implementations.

6.1 Model Deployment Tools

This subsection explores a range of lightweight deployment toolchains designed for resource-constrained edge devices, tracing their evolution from early C++ converters to modern platforms with support for quantization, AutoML, and hardware-specific integration. Early frameworks such as uTensor [67] demonstrated feasibility by converting TensorFlow models into C++ code but lacked support for quantization or advanced operators. TFLite Micro addressed these limitations by adding support for 8-bit quantization, expanded operator coverage, and community backing, making it a de facto baseline in TinyML deployments [68]. However, TFLite Micro requires manual tuning and has no built-in GUI. To lower entry barriers, Edge Impulse [69] introduced a no-code AutoML

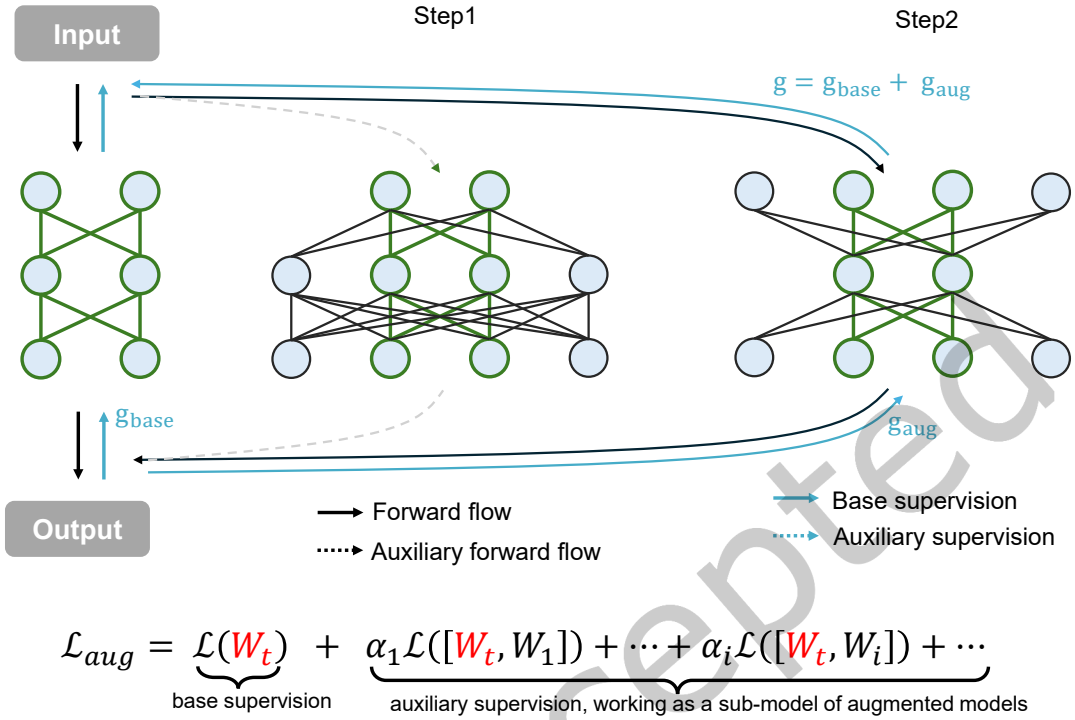

Fig. 5. Network augmentation strategy: A tiny model is trained within a larger model to benefit from auxiliary supervision, but only the tiny network is used during inference [66].

pipeline with integrated DSP processing and on-device testing. TFLite Model Maker [70] supports fine-tuning and exporting models tailored for deployment on EdgeTPUs and mobile hardware. PyTorch Mobile [71], while not suitable for MCUs, supports deployment of larger TinyDL models (including Transformers) on higher-end mobile SoCs.

More specialized toolchains target performance tuning and low-level integration. CMSIS-NN [72] provides hand-optimized kernels for ARM Cortex-M architectures and is often paired with TFLite Micro for improved inference latency. MicroTVM [73], as an extension of the TVM compilation stack, brings auto-tuning and graph optimization to MCU platforms like STM32 and ESP32. Glow [74], developed by Meta, offers ahead-of-time graph lowering for hardware accelerators and NPUs. Tools like DeepC convert Keras models into static C code, ideal for systems without dynamic memory support [75], while MLPACK [76], although not TinyML-specific, is a lightweight C++ library adaptable for embedded use. Vendor-specific tools such as X-CUBE-AI [77] for STM32 platforms and academic solutions like QKeras with HLS4ML [78] for FPGA deployment demonstrate the growing ecosystem of domain-targeted solutions. Commercial platforms such as OctoML [79] and Nebullvm [80] further enhance deployment by automating compilation, quantization, and precision tuning across edge platforms. These diverse tools reflect the increasing demand for streamlined, hardware-aware TinyDL deployment pipelines.

6.2 Compilation and Runtime Support

To execute models efficiently on tiny hardware, compilation and runtime libraries play a vital role. They optimize memory usage, operator execution, and compatibility with various MCUs, DSPs, and NPUs. CMSIS-NN [72] was

one of the earliest contributions in this space, providing hand-optimized fixed-point kernels for ARM Cortex-M chips. Although it does not support automatic graph compilation, it is frequently paired with TFLite Micro to yield significant gains in latency and power efficiency. To move beyond manually crafted kernels, frameworks such as TVM [81] introduced automated compilation workflows that include graph-level optimization, operator fusion, and quantization-aware tuning. Its extension, MicroTVM [73], enables deployment on bare-metal devices such as STM32 and ESP32, with support for auto-tuning and memory planning specific to target hardware.

Other frameworks have pursued alternative compilation strategies. Glow [74], developed by Meta, lowers ML graphs into intermediate representations and compiles them for hardware accelerators, offering static deployment advantages on NPUs. Accelerated Linear Algebra [82], originally part of TensorFlow, generates backend-specific binaries through operation fusion, and although primarily used in server environments, its techniques influence edge compiler design. ONNX Runtime [83], when paired with execution providers like TensorRT and OpenVINO, facilitates optimized deployment on edge GPUs and NPUs, though its MCU support remains limited. Emerging tools such as Apache NNC and Tensor Comprehensions [84] explore polyhedral compilation and domain-specific optimization, primarily in research settings. Commercial efforts like OctoML [79] offer automated TVM-based tuning services, while Nebullvm [80] supports quantization, pruning, and cross-platform model optimization. Together, these compilation frameworks ensure that deep learning models, once compressed and optimized, can meet the strict memory and latency constraints of TinyML deployments.

6.3 End-to-End Platforms

This section highlights the critical role of compilation and runtime libraries in executing machine learning models efficiently on tiny edge hardware, focusing on tools that enable memory optimization, operator tuning, and hardware-specific acceleration across MCUs, DSPs, and NPUs.

While deployment and compilation tools handle isolated phases of the TinyML lifecycle, end-to-end platforms offer unified environments, with some prioritizing usability and others pushing optimization boundaries. These platforms vary in abstraction, hardware support, and model adaptability, often building on or compensating for the limitations of one another. This section offers an overview of end-to-end TinyML platforms that streamline the entire machine learning lifecycle for edge deployment, while Table 4 provides a detailed comparison of toolchains and frameworks based on platform support, compression, quantization, AutoML capabilities, and community maturity.

Beyond these widely used frameworks, several specialized platforms provide tailored functionalities that address the needs of different domains. TensorFlow Lite Model Maker [70] offers a Python-based AutoML pipeline with fine-tuning, pruning, and quantization support for deployment on mobile and EdgeTPU devices. For time-series and sensor data, platforms such as Qeexo AutoML [25] and SensiML [28] offer end-to-end workflows that integrate feature extraction, model compression, and deployment to MCUs and FPGAs. These frameworks are especially suited for industrial and automotive applications requiring vibration or IMU-based inference.

To support optimization-focused deployment, platforms such as Latent AI's LEIP stack [86] and OctoML [79] emphasize model compression workflows across CPUs, NPUs, and MCUs. These frameworks integrate mixed-precision quantization, pruning, and tuning for performance–energy trade-offs. While OctoML builds upon the TVM stack for deployment optimization, Latent AI is designed for hybrid edge use cases requiring cross-platform support.

Table 4. Comparison of TinyML Toolchains and Frameworks

Toolchain / Framework	Supported Platforms	Compression Support	Quantization Support	NAS / AutoML Support	Community Maturity / Ecosystem
TensorFlow Lite Micro [68]	MCU (ARM Cortex-M), Arduino, ESP32	Pruning, weight clustering	8-bit, int4 (experimental)	Manual only	Very high; large community, strong docs
uTensor [67]	ARM Cortex-M, STM32	Minimal	8-bit only	No	Low; legacy status, low activity
CMSIS-NN [72]	ARM Cortex-M family	Manual optimizations	8-bit fixed-point	No	Medium; well-documented for ARM devs
Edge Impulse Studio [69]	Web-based IDE (ESP32, STM32, Arduino)	Pruning + DSP fusion	Quant-aware training	Yes (AutoML built-in)	High; growing ecosystem, no-code UI
MicroTVM [73]	RISC-V, ARM, x86, ESP32	Compiler-based optimization	INT8/INT4	Yes (TVM autotuning)	High among compiler researchers
TFLite Model Maker [70]	Android, Raspberry Pi, EdgeTPU	Basic pruning	8-bit, int16	Yes (UI-based)	High for mobile/EdgeTPU apps
Qeexo AutoML [25]	ARM Cortex-M, IMU boards	Auto-selected models	Yes (auto)	Yes (end-to-end)	Medium; strong for sensor ML
Neuton TinyML [85]	TinyMCU (<1 KB RAM)	Highly compressed models	Yes (proprietary)	Yes (Auto-compression)	Emerging; niche but focused
Latent AI [86]	MCU, CPU, NPU	Pruning, quantization, distillation	Yes (mixed-precision)	Yes (LEIP AutoML)	Medium; commercial/enterprise use
SensiML [28]	QuickLogic FPGA, MCUs	Auto feature selection	Yes (auto-tuned)	Yes	Growing in sensor-specific domains
OctoML [79]	Cloud, Edge (TVM-compatible)	TVM-based optimization	Yes (via TVM)	TVM-integrated AutoTuner	High in TVM community, enterprise users
Arduino IDE / ArduinoML [87]	Arduino boards (AVR, Cortex-M)	None	8-bit (manual)	No	Large maker community
NXP eIQ Toolkit [88]	NXP MCUs and NPUs	Quantization, pruning (NXP SDK)	Yes	Limited (GUI only)	Vendor-supported; stable in NXP flow
Microsoft EdgeML [89]	MCU, DSPs (research only)	Model compression (Bonsai, ProtoNN)	Yes	No	Academic; low deployment focus
Google Colab + TFLite [90]	Cloud to TFLite-compatible edge	Manual via TFLite converter	Yes	Manual	Broad TF support; no GUI
Sony AI Studio [91]	Sony Spresense board	Pre-configured	Yes (limited)	Model Zoo only	Specialized; Sony-specific
KaaEdge AI	Edge devices, IoT networks	Model coordination only	Depends on device	Federated AutoML (in development)	System-level orchestration; growing

Popular TinyML Deployment Toolchains Compared

TensorFlow Lite Micro: de-facto baseline; 8-bit INT quantization, huge community; no GUI [68].
Edge Impulse Studio: drag-and-drop AutoML with built-in DSP blocks-ideal for newcomers [69].
MicroTVM: TVM-based compiler autotuning for MCUs (STM32, ESP32) and RISC-V boards [73].
Neuton TinyML: sub-kilobyte models that fit where even CMSIS-NN is too heavy [85].

Other tools address the needs of beginner users or are integrated within hardware-specific ecosystems. Arduino IDE and ArduinoML [87] offer intuitive interfaces for model deployment on AVR and Cortex-M boards but are limited to simpler use cases. The NXP eIQ Toolkit [88] provides vendor-specific integration for NXP's MCU and NPU portfolio, offering a graphical interface with built-in support for quantization and pruning.

From the research perspective, Microsoft's EdgeML [89] introduces novel model architectures such as ProtoNN [92] for ultra-low-memory inference. For advanced users, the Google Colab and TFLite workflow [90] provides maximum scripting flexibility, allowing users to train models in the cloud and convert them for deployment using the TFLite converter. Sony AI Studio [91], while limited to the Spresense board, offers a curated development environment for vision and audio inference.

Together, these platforms span a wide spectrum-from GUI-based environments to optimization-first toolchains-offering developers diverse options depending on their expertise, application complexity, and hardware targets.

7 Applications of TinyML and TinyDL

TinyML and TinyDL are enabling real-time, low-power, and privacy-aware AI solutions across a range of domains by embedding intelligence directly into edge devices. This section outlines key applications in vision, audio and language processing, healthcare, and industrial-environmental monitoring.

7.1 Vision

TinyML is advancing computer vision tasks such as small object detection (SOD) under hardware constraints through super-resolution, image pyramids, and feature pyramid networks [93]. Generative Adversarial Networks and log-gradient inputs improve image quality and energy efficiency on constrained devices [94]. New benchmarks like Wake Vision support person detection [95], while lightweight models like YoLite+ enable fast, multi-object detection in traffic scenarios [96]. These methods significantly expand edge deployment feasibility for vision tasks.

7.2 Audio and Natural Language Processing

TinyML supports low-power applications such as keyword spotting, speech recognition, and environmental sound monitoring on devices like the Arduino Nano BLE Sense [97]. Model compression, Fast Fourier Transform (FFT)-based pipelines, and privacy-aware approaches like split learning enhance usability for on-device NLP tasks such as sentiment analysis and behavior modeling [98, 99]. Edge-suitable language models, including TinyBERT, are deployed using pruning, quantization, and simplified architectures [100]. Recent systems like TinyChirp showcase real-time bird sound classification on wireless acoustic sensors [101].

7.3 Healthcare and Human Behavior Analytics

TinyML is increasingly deployed in healthcare for ECG and respiratory signal monitoring using low-power microcontrollers [102, 103]. These applications support continuous diagnosis and feedback with enhanced privacy and lower latency. Additionally, emotion recognition and Human Activity Recognition (HAR) benefit from wearable sensors, photoplethysmography, EEGs, and mmWave radar [104, 105]. CNN and LSTM-based models run locally to reduce cloud dependency, while wristband-based HAR systems offer real-time classification and improved energy efficiency [106].

7.4 Industrial and Environmental Applications

TinyML supports predictive maintenance and anomaly detection in industrial systems by analyzing audio, vibration, and thermal data directly on the edge [107, 108]. Deeply quantized and binary models improve performance under strict latency and memory budgets [109]. For environmental applications, TinyML enables

smart agriculture through edge-native disease detection systems like the Nuru app [110], while bioacoustic and wearable sensors support wildlife tracking in remote regions [111, 112]. Integration with blockchain and federated learning further enhances security and transparency in industrial IoT systems [113, 114].

TinyML and TinyDL continue to expand the boundaries of intelligent edge computing, enabling robust AI capabilities across constrained real-world environments.

8 On-Device Learning and Reformability

The evolution of TinyML has opened new frontiers for executing artificial intelligence directly on resource-constrained endpoint devices [9, 115]. Indeed, a significant paradigm shift within this domain is the move from static, pre-trained models toward dynamic systems capable of learning and adapting post-deployment [8]. This progression towards on-device learning and the concept of reformability is critical for the long-term viability and effectiveness of TinyML applications, particularly as they become integrated into dynamic, real-world environments [15, 27]. Such capabilities not only enhance model performance over time but also offer substantial benefits for user privacy and data security by keeping sensitive information localized on the device [19, 116].

8.1 Continual and Few-Shot Learning on MCUs

Traditional TinyML workflows involve training a machine learning model offline on powerful servers and then deploying a highly optimized, static version for inference on a MCU [115]. However, this approach is limited, as the performance of a static model can degrade when the data distribution it encounters in the real world changes over time, a phenomenon known as concept drift [8]. To overcome this limitation, research has increasingly focused on enabling learning capabilities directly on the MCU. A promising solution is the implementation of online or continual learning, where the model can be updated incrementally as new data becomes available. This allows the device to adapt to changing environmental conditions without requiring a complete redeployment of the model [117]. For instance, the TinyOL framework was proposed to facilitate online learning on MCUs, allowing them to learn from a continuous stream of data [8]. This approach is instrumental for applications that require long-term autonomy and robustness. Furthermore, advancements in few-shot learning are enabling MCUs to train effectively on a very small number of examples. This is particularly relevant for customization and personalization, where a device might need to learn new keywords or commands specific to a user. Research in few-shot KWS has demonstrated that models deployed on MCUs can be trained to recognize new words with only a handful of training samples, making the end-user experience significantly more flexible and interactive [118]. Such on-device training capabilities are often facilitated through methods like federated learning, which allows for collaborative model training across multiple decentralized devices while keeping the raw data localized [119].

8.2 Reformable TinyML

Building upon the concept of on-device learning is the emerging paradigm of Reformable TinyML. This refers to a holistic framework where TinyML systems are engineered with the inherent capability to be modified, updated, or reformed after their initial deployment [15]. The primary goal is to create resilient and sustainable intelligent systems that can self-diagnose performance degradation and trigger an adaptation process to maintain accuracy and efficiency over their operational lifetime [27]. The reformable pipeline extends beyond simple on-device updates. It encompasses a structured workflow that includes monitoring for data drift, evaluating the current model's efficacy, and invoking a reformation strategy when necessary. This strategy could involve on-device fine-tuning, fetching a model patch from an edge server, or participating in a federated learning task to receive an updated global model [15]. Consequently, reformability ensures that the intelligence at the extreme edge does

not become obsolete, thereby enhancing the overall value and reliability of the IoT ecosystem. The architecture of such a pipeline is conceptually illustrated in Figure 6.

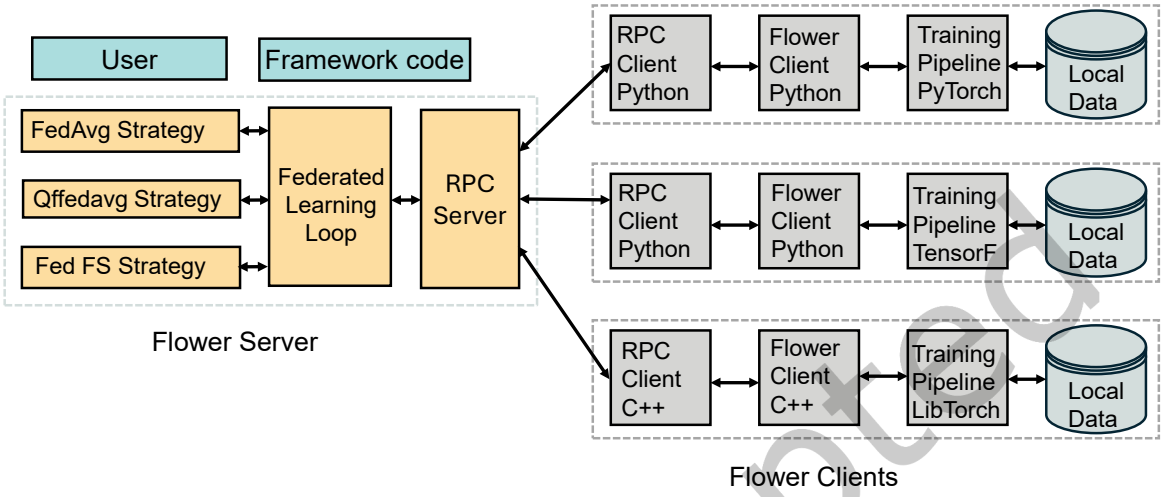

Fig. 6. Architecture for Privacy-Preserving Federated TinyML [15]

8.3 Privacy and Security Benefits

A foundational advantage of processing data at the edge is the inherent enhancement of privacy and security [115]. By performing ML inference directly on the MCU, sensitive user data, such as audio from a microphone or images from a camera, does not need to be transmitted to the cloud or a remote server [9]. This localization of data significantly reduces the risks associated with data breaches during transmission and unauthorized access on third-party servers, a crucial consideration for human-centered applications [19]. The push for on-device learning and reformability further strengthens these privacy guarantees. In a federated learning context, for example, the raw data used for training never leaves the user's device. Instead, only anonymized model updates or gradients are shared to contribute to a global model, ensuring collaborative learning without compromising individual privacy [120]. Moreover, recent work has focused on integrating explicit privacy-preserving mechanisms into the TinyML framework. Techniques such as local differential privacy can be implemented to add statistical noise to data or model updates before they are shared, providing mathematical guarantees of privacy [121]. In conjunction with secure hardware features like memory protection units (MPUs) or secure enclaves to encrypt model parameters, these methods provide a robust, multi-layered approach to securing intelligent edge devices [121].

9 Evaluation Metrics and Benchmarks

Evaluating TinyML models requires a holistic framework that addresses the unique constraints of ultra-resource-constrained environments while preserving sufficient task performance for real-world deployment. Unlike conventional machine learning systems evaluated primarily based on accuracy or F1-score, TinyML models must be evaluated using a blend of metrics that reflect efficiency, scalability, deployment feasibility, and trade-offs between model performance and resource utilization. These metrics span a wide spectrum, including computational efficiency (inference latency, throughput), memory and storage footprint (model size, RAM usage),

energy consumption (power and battery profile), and task-specific accuracy under quantized or pruned conditions. Moreover, consistent comparison across approaches is facilitated by a small set of curated benchmarks and datasets tailored for the TinyML domain, such as KWS, low-resolution vision tasks, and sensor signal classification.

9.1 Efficiency and Compression Metrics

In TinyML, efficiency is a first-class citizen. Edge devices such as ARM Cortex-M MCUs or RISC-V-based embedded systems typically offer only kilobytes of RAM and flash storage, no floating-point hardware, and must run on energy-constrained or even battery-less systems. Consequently, performance metrics in TinyML revolve around the ability to deploy models that fit within these extreme limitations without sacrificing critical functionality. The most frequently used efficiency metrics include:

9.1.1 Model Size (KB). Model size, measured in kilobytes, represents the total footprint of the model when stored on the device, including weights, biases, and optionally additional metadata. Given that popular MCUs (e.g., STM32F746) have only 512 KB to 1 MB of flash memory, model size is often capped at 250–500 KB to allow room for the operating system and runtime libraries. Compression techniques such as weight pruning, Huffman encoding, and PTQ are employed to reduce model size without significant degradation in performance. For instance, Han et al. [122] demonstrate that deep neural networks can be compressed by 9–13× using a combination of pruning and quantization, with minimal accuracy drop. MCUNet [39], an architecture designed specifically for MCUs, achieves ResNet-level accuracy on ImageNet with models as small as 300 KB.

9.1.2 Inference Latency (ms). Inference latency is the wall-clock time taken to perform one forward pass of the model on the target hardware, typically measured in milliseconds. Latency affects the responsiveness of real-time systems like gesture recognition or wake-word detection. For example, Google's KWS model for TensorFlow Lite Micro executes in less than 20 ms on a Cortex-M4 at 80 MHz [123]. Latency is influenced by model depth, width, and data precision (e.g., 8-bit vs. floating-point), and can be reduced using techniques like operator fusion, loop unrolling, and hardware-specific optimizations such as CMSIS-NN [72].

9.1.3 Memory Usage (KB). Memory usage includes both the *static memory footprint* (the size of the model parameters) and *dynamic memory* (temporary activations, intermediate buffers). MCUs typically have only tens to hundreds of kilobytes of SRAM; for example, the STM32L475 has just 128 KB of SRAM, which limits the size of intermediate tensors and imposes constraints on model depth and width [123]. Memory profiling tools in frameworks such as TensorFlow Lite Micro [68] and TVM [124] provide developers with visibility into memory allocation and enable optimization of memory usage across inference stages. Techniques such as activation recomputation (also known as gradient checkpointing) [125] and in-place memory reuse [126] are particularly effective in minimizing peak RAM usage. Additionally, CMSIS-NN [72] and other hand-optimized libraries help reduce memory overhead by using fixed-point arithmetic and optimizing buffer allocation for low-latency execution on Arm Cortex-M processors.

9.1.4 Operations Count (MACs / FLOPs). Operations count refers to the total multiply-accumulate operations (MACs) or floating-point operations (FLOPs) required for a single inference. It is often used as a proxy for computational workload and correlates strongly with both latency and energy consumption on MCU-class hardware. MLPerf Tiny includes MACs as a primary metric to standardize comparisons across architectures, enabling fair trade-off analysis across accuracy, latency, and energy [31].

9.1.5 Energy Consumption (mJ). For battery-powered or energy-harvesting devices, energy efficiency is paramount. Energy per inference, measured in millijoules (mJ), is a product of inference latency and average power consumption. Banbury et al. [31] show that their benchmark models consume between 0.1 mJ and 10 mJ per

inference. Specialized accelerators like the GAP8 or Ambiq Apollo chips allow sub-mJ inference for tasks like image classification or speech recognition.

9.2 Accuracy Trade-offs and Constraints

Accuracy remains an important performance metric, particularly when TinyML is used in critical applications such as medical diagnostics or industrial anomaly detection. However, due to severe memory and compute limitations, models are often subject to trade-offs between efficiency and accuracy. One of the most widely studied compromises is the balance between quantization and model performance.

9.2.1 Quantization vs. Accuracy. Quantization reduces the precision of weights and activations, typically from 32-bit floating point (FP32) to 8-bit integers (INT8), and sometimes lower. This can lead to massive reductions in model size, latency, and energy usage, but may affect model accuracy. Magnitude-based structured pruning of MobileNetV2 (50% weights removed) incurs < 1% top-1 drop on ImageNet while shrinking model size by 1.9 times [62]. Jacob et al. [127] found that PTQ can reduce MobileNetV1 accuracy on ImageNet by up to 3–4%, though QAT can mitigate this. QAT introduces fake quantization nodes during training, allowing the network to compensate for precision loss. Special training procedures and loss-aware quantization [128] are employed for binary networks. Beyond QAT and PTQ, several quantization schemes have emerged to address different levels of granularity and trade-off. For instance, per-channel quantization adjusts the scale and zero-point for each output channel, leading to better numerical stability and often improved accuracy compared to per-tensor quantization [129]. Mixed-precision quantization allows different layers or operations to use different bit-widths (e.g., 8-bit for early layers and 4-bit for later layers), striking a balance between efficiency and performance [130]. Techniques like DoReFa-Net [131] and Learned Step Size Quantization [132] further improve accuracy by learning optimal quantization parameters during training. In the TinyML context, these methods have enabled the deployment of accurate models like ResNet and MobileNet variants on MCUs with under 256 KB of SRAM. Furthermore, hardware-aware quantization strategies are increasingly integrated into deployment pipelines using tools such as TensorFlow Model Optimization Toolkit [133] and PyTorch's quantization API, enabling automated conversion and validation across platforms.

9.2.2 Constraints Beyond Quantization. TinyML models often face deployment-specific constraints such as memory budgets (e.g., 64 KB of RAM), latency caps (e.g., 10 ms), and energy limits (e.g., 1 mJ per inference). Frameworks like μNAS [134] and Once-for-All (OFA) [135] support constraint-aware NAS to generate tailored models. In practice, trade-offs are carefully evaluated to balance latency, accuracy, and reliability. Constraint-aware modeling goes beyond architectural choices and encompasses compiler-level and deployment-time optimizations. For example, models must comply with quantization compatibility constraints of hardware accelerators like the GAP8 SoC, which supports only INT8 convolutions [136], or the Ambiq Apollo3 Blue, which requires careful SRAM and DMA management to maintain sub-mW operation [137]. Real-time constraints also vary across application domains, such as, voice-triggered devices may tolerate 10–20 ms of latency, whereas anomaly detection in industrial sensors may allow hundreds of milliseconds, but must operate within a strict energy envelope. To handle such variance, modern compilers such as TVM [124], Glow [74], and Apache Relay perform cross-layer optimization and memory layout transformations that respect such deployment constraints. Additionally, tools like MCUNetV2 [138] integrate NAS with firmware-level profiling to co-optimize models for specific MCUs, achieving a better trade-off across the energy-latency-accuracy spectrum.

9.3 Standard Datasets and Benchmarks

To enable reproducibility and fair comparison across TinyML approaches, the community relies on a curated set of datasets and benchmarks reflecting the domain's constraints and typical use cases. These benchmarks target

tasks such as KWS, low-resolution vision, and anomaly detection. Table 5 summarizes the standard benchmark datasets widely adopted in the TinyML community, covering representative tasks such as keyword spotting, low-resolution image classification, and lightweight visual wake-word detection.

Table 5. Summary of benchmark datasets and their characteristics

Dataset	Task	Input Size	Classes	Use Case
Google Speech Commands [139]	Keyword spotting	1 s audio (16 kHz)	12–35	Voice command detection
Visual Wake Words [38]	Person detection	96×96 RGB	2	Vision-triggered wake-up
Tiny ImageNet [140]	Image classification	64×64 RGB	200	Low-resolution object recognition
CIFAR-10/100 [141]	Image classification	32×32 RGB	10/100	Lightweight vision tasks
μMLPerf [36]	Benchmark suite	Various	Various	Standardized TinyML evaluation

9.3.1 Google Speech Commands. Developed by Warden et al., this dataset contains short spoken words sampled at 16 kHz. It is the standard benchmark for KWS. Tasks involve recognizing a fixed vocabulary such as "*yes*", "*no*", and "*go*".

9.3.2 Visual Wake Words. This dataset is a binary classification task for detecting the presence of a person in low-resolution images. It is used in wake-word-style visual triggers for cameras or embedded vision systems.

9.3.3 Tiny ImageNet and CIFAR. These datasets serve as benchmarks for image classification under low-resolution and low-memory conditions. Tiny ImageNet is more challenging due to its 200-class design, while CIFAR remains widely used for comparison.

9.3.4 μMLPerf Benchmark Suite. MLCommons introduced μMLPerf to provide standardized evaluation across KWS, image classification, and anomaly detection. It includes metrics like accuracy, model size, memory footprint, and energy per inference, making it one of the most comprehensive benchmarks for TinyML systems.

10 Challenges and Open Research Problems

Despite its rapid progress, TinyDL still faces fundamental challenges that must be addressed to achieve reliable, secure, and generalizable edge intelligence [56, 142]. A persistent challenge is the trade-off between model accuracy and resource efficiency. TinyDL systems must operate within severe memory and compute constraints, often limiting model size to under 1MB and compute budgets to sub-milliwatt levels. Achieving high accuracy within such limits remains difficult, particularly for complex tasks like image classification or NLP [143]. Additionally, the memory hierarchy of typical MCUs—featuring limited SRAM (e.g., 320KB) and constrained flash storage—complicates not only model storage but also intermediate activation handling during inference. Optimizing memory allocation across the entire network rather than individual layers is essential for efficient deployment [143].

Computational bottlenecks further restrict the deployment of state-of-the-art deep learning architectures. Transformer models, with their quadratic complexity in attention mechanisms, are especially challenging to run on low-power devices. While recent innovations like sparse attention and approximate self-attention mechanisms offer potential, they remain computationally demanding for real-time execution on microcontrollers [61]. Moreover, few existing toolchains provide optimized attention primitives or support for layer normalization, limiting the practical deployment of compact transformer variants like TinyBERT or DistilBERT on devices such as ARM Cortex-M4 [61].

Security is another critical concern, especially given the physical accessibility of edge devices in uncontrolled environments. Adversarial examples generated on powerful machines can transfer effectively to TinyML models

deployed on devices like ESP32 and Raspberry Pi, making them vulnerable to extraction and evasion attacks [144, 145]. Additionally, secure model updates remain a challenge due to the limited computational capacity for cryptographic validation. Although hardware security modules and trusted execution environments offer potential defenses, they introduce cost and power overheads that conflict with the goals of ultra-low-power AI [116].

Supporting adaptability through on-device learning introduces another layer of difficulty. TinyML devices often require personalization using only a few labeled examples, such as in health monitoring wearables. However, few-shot learning methods generally rely on meta-learning algorithms that exceed MCU memory and compute capabilities [146, 147]. Similarly, continual learning techniques must avoid catastrophic forgetting while learning from limited new data. Current methods require memory-intensive replay strategies or complex parameter regularization, neither of which are optimized for low-resource deployment [148].

Benchmarking remains inconsistent across the TinyDL ecosystem. Unlike cloud ML, TinyDL requires evaluation beyond accuracy, incorporating latency, energy consumption, model size, and memory footprint. While MLPerf Tiny provides a useful starting point, it is limited in scope—focusing largely on image and audio tasks while overlooking domains such as natural language understanding and multimodal sensor fusion [31].

Finally, the TinyDL software ecosystem still lacks mature, portable toolchains capable of supporting advanced deep learning models across diverse platforms. Tool support for key operations—such as attention mechanisms or mixed-precision arithmetic—is often inconsistent, requiring developers to handcraft model variants for specific targets. This lack of cross-platform compatibility increases development overhead and hinders the scalability of TinyDL deployments across heterogeneous hardware environments [143]. Addressing these open challenges will require cross-disciplinary innovation across machine learning, systems architecture, compiler design, and embedded security. Solving them will be critical to unlocking the full potential of TinyDL in real-world, high-impact edge applications [149, 150].

11 Future Directions

As TinyDL systems mature and expand across diverse application domains from healthcare and smart homes to industrial automation and autonomous sensing, new challenges and technological frontiers are emerging. Addressing these will require interdisciplinary advances in hardware design, algorithmic efficiency, secure training, and adaptable software ecosystems. This section outlines five promising directions for future exploration. Neuromorphic architectures employing Spiking Neural Networks offer an alternative computational paradigm designed for ultra-low-power, event driven processing typical of brain-inspired systems. These architectures promise efficient always-on inference on MCU-scale devices, ideal for continuous monitoring applications-with hardware platforms like BrainChip's Akida and Intel's Loihi leading the way. To fully leverage Spiking Neural Networks in TinyDL, research must advance surrogate-gradient training, event encoding techniques, and software toolchains that map spiking models onto neuromorphic hardware seamlessly [62].

Implementing federated learning in TinyDL contexts addresses privacy and adaptability by enabling decentralized learning across devices without sharing raw data crucial for distributed sensor networks. Lightweight frameworks such as TinyFedTL and TinyMetaFed demonstrate on-device aggregation of quantized updates, yet challenges remain in managing communication overhead, heterogeneous device capabilities, and adversarial resilience [145]. Future work must focus on sparsified updates, asynchronous or hierarchical FL protocols, and secure aggregation mechanisms amenable to TinyML constraints.

Tiny Foundation Models refer to miniaturized versions of large pretrained models intended for deployment on edge hardware. Promising techniques such as knowledge distillation, structured pruning, and quantizationapplied to models like TinyViT have shown the potential to reduce model size to MCU-suitable scales while preserving task performance [39, 60]. The next step is to enable modular foundational architectures, where a general "backbone"

pre-trained model supports multiple lightweight task-specific heads, with workflows powered by on-device or Edge AutoML-enabled fine-tuning. Edge AutoML seeks to automate the process of designing, compressing, and deploying TinyDL models on resource-constrained devices. Techniques like hardware-aware NAS frameworks, such as TinyNAS and Once-for-All Networks, have demonstrated effective ways to balance accuracy with memory and latency constraints [58, 61]. However, integrating AutoML into full deployment pipelines remains an open challenge. Future research should focus on combining AutoML with model compression strategies like quantization and pruning and incorporating hardware feedback to generate models that are not only accurate but also energy-efficient and deployable in real-world TinyDL scenarios.

Domain-specific accelerators, including NPUs, ASICs, FPGAs, and specialized RISC-V engines, offer substantial gains in inference speed, energy efficiency, and model scalability for TinyDL. Devices like the EdgeTPU and transformer-focused RISC-V extensions efficiently deliver quantized convolution and attention workloads, outperforming general-purpose MCUs [61, 62]. The challenge now is to develop advanced compilation toolchains that partition and schedule TinyDL models across heterogeneous hardware, integrate with platforms like TFLite Micro and CMSIS-NN, and maximize runtime configuration flexibility without sacrificing portability or ease of development [73, 74].

12 Conclusions

This survey presents a comprehensive examination of the evolution from TinyML to TinyDL, highlighting how the convergence of efficient model architectures, software toolchains, and hardware platforms has enabled sophisticated on-device intelligence in severely resource-constrained environments. We begin by delineating the scope and distinction between TinyML and TinyDL, emphasizing the growing need to embed deep learning capabilities, once reserved for data centers, into low-power MCUs and edge devices. We have outlined the hardware advancements, including the emergence of neural accelerators and specialized ASICs, that now support the deployment of deep networks with kilobyte-scale memory footprints and milliwatt power budgets. Simultaneously, we explored the critical role of model optimization techniques such as quantization, pruning, and joint compression strategies, as well as the contributions of NAS in tailoring architectures to edge constraints. On the software side, we cataloged an extensive range of deployment frameworks, compiler toolchains, and AutoML platforms that streamline the end-to-end TinyDL lifecycle. Through domain-specific applications in vision, audio, healthcare, and industrial monitoring, we demonstrated the transformative potential of TinyDL across sectors demanding low latency, energy efficiency, and data privacy.

Looking ahead, TinyDL is poised to catalyze a new generation of edge-native intelligence. This includes the development of neuromorphic architectures using spiking neural networks, federated learning for decentralized personalization, and ultra-lightweight foundation models capable of generalization across tasks and modalities. The co-design of hardware and software will become increasingly central, as will the creation of standardized, energy-aware benchmarks to evaluate system performance holistically. By bridging the conceptual, architectural, and practical aspects of TinyDL, this survey aims to serve as a foundational resource for both researchers and practitioners. It underscores the critical shift from cloud dependence to autonomous, efficient edge intelligence, laying the groundwork for continued innovation in AI at the very edge of computing.

References

[1] Syed Ali Raza Zaidi, Ali M Hayajneh, Maryam Hafeez, and Qasim Zeeshan Ahmed. 2022. Unlocking edge intelligence through tiny machine learning (TinyML). *IEEE Access* 10 (2022), 100867–100877.

[2] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, and Song Han. 2023. Tiny machine learning: Progress and futures [feature]. *IEEE Circuits and Systems Magazine* 23, 3 (2023), 8–34.

[3] Hui Han and Julien Siebert. 2022. TinyML: A systematic review and synthesis of existing research. In *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 269–274.

[4] Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. 2017. TinyDL: Just-in-time deep learning solution for constrained embedded systems. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.

[5] Minh Tri Lê, Pierre Wolinski, and Julyan Arbel. 2023. Efficient neural networks for tiny machine learning: A comprehensive review. *arXiv preprint arXiv:2311.11883* (2023).

[6] Jihong Park, Sumudu Samarakoon, Mehdi Bennis, and Mérouane Debbah. 2019. Wireless network intelligence at the edge. *Proc. IEEE* 107, 11 (2019), 2204–2239.

[7] Stanislava Soro. 2021. TinyML for ubiquitous edge AI. *arXiv preprint arXiv:2102.01255* (2021).

[8] Youssef Abadade, Anas Temouden, Hatim Bamoumen, Nabil Benamar, Yousra Chtouki, and Abdelhakim Senhaji Hafid. 2023. A comprehensive survey on tinyml. *IEEE Access* 11 (2023), 96892–96922.

[9] Partha Pratim Ray. 2022. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University-Computer and Information Sciences* 34, 4 (2022), 1595–1623.

[10] Danilo Pau and Prem Kumar Ambrose. 2022. Automated neural and on-device learning for micro controllers. In *2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON)*. IEEE, 758–763.

[11] Giovanni Delnevo, Silvia Mirri, Catia Prandi, and Pietro Manzoni. 2023. An evaluation methodology to determine the actual limitations of a tinyml-based solution. *Internet of Things* 22 (2023), 100729.

[12] Luigi Capogrosso, Federico Cunico, Dong Seon Cheng, Franco Fummi, and Marco Cristani. 2024. A machine learning-oriented survey on tiny machine learning. *IEEE Access* 12 (2024), 23406–23426.

[13] Lina Bariah, Qiyang Zhao, Hang Zou, Yu Tian, Faouzi Bader, and Merouane Debbah. 2024. Large generative ai models for telecom: The next big thing? *IEEE Communications Magazine* 62, 11 (2024), 84–90.

[14] Imopishak Thingom and N Basanta Singh. 2023. A Review on Machine Learning in IoT Devices. *International Journal of Digital Technologies* 2, 1 (2023).

[15] Visal Rajapakse, Ishan Karunanayake, and Nadeem Ahmed. 2023. Intelligence at the extreme edge: A survey on reformable TinyML. *Comput. Surveys* 55, 13s (2023), 1–30.

[16] Nasser Alajlan and Dalia M. Ibrahim. 2022. TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications. *Micromachines* 13, 6 (2022), 851. DOI: http://dx.doi.org/10.3390/mi13060851

[17] Abdussalam Elhanashi, Pierpaolo Dini, Sergio Saponara, and Qinghe Zheng. 2024. Advancements in TinyML: Applications, Limitations, and Impact on IoT Devices. *Electronics* 13, 17 (2024), 3562.

[18] Georgios Kornaros. 2022. Hardware-assisted machine learning in resource-constrained IoT environments for security: review and future prospective. *IEEE Access* 10 (2022), 58603–58622.

[19] Ismail Lamaakal, Ibrahim Ouahbi, Khalid El Makkaoui, Yassine Maleh, Paweł Pławiak, and Fahad Alblehai. 2024. A TinyDL model for gesture-based air handwriting Arabic numbers and simple Arabic letters recognition. *IEEE Access* (2024).

[20] Zeinab E Ahmed, Aisha A Hashim, Rashid A Saeed, and Mamoon M Saeed. 2024. TinyML network applications for smart cities. In *TinyML for Edge Intelligence in IoT and LPWAN Networks*. Elsevier, 423–451.

[21] Norah N Alajlan and Dina M Ibrahim. 2024. Original Research Article TinyML: Adopting tiny machine learning in smart cities. *Journal of Autonomous Intelligence* 7, 4 (2024).

[22] Ivan Khokhlov, Egor Davydenko, Ilya Osokin, Ilya Ryakin, Azer Babaev, Vladimir Litvinenko, and Roman Gorbachev. 2020. Tiny-YOLO object detection supplemented with geometrical data. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 1–5.

[23] Nithesh Singh Sanjay and Ali Ahmadinia. 2019. MobileNet-Tiny: A deep neural network-based real-time object detection for rasberry Pi. In *2019 18th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 647–652.

[24] Brett Koonce. 2021. SqueezeNet. In *Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization*. Springer, 73–85.

[25] Riku Immonen and Timo Hämäläinen. 2022. Tiny Machine Learning for Resource-Constrained Microcontrollers. *Journal of Sensors* 2022, 1 (2022), 7437023.

[26] Danilo Pau, Abderrahim Khiari, and Davide Denaro. 2021. Online learning on tiny micro-controllers for anomaly detection in water distribution systems. In *2021 IEEE 11th International Conference on Consumer Electronics (ICCE-Berlin)*. IEEE, 1–6.

[27] Rakhee Kallimani, Krishna Pai, Prasoon Raghuwanshi, Sridhar Iyer, and Onel LA López. 2024. TinyML: Tools, applications, challenges, and future research directions. *Multimedia Tools and Applications* 83, 10 (2024), 29015–29045.

[28] Swapnil Sayan Saha, Sandeep Singh Sandha, and Mani Srivastava. 2022. Machine learning for microcontroller-class hardware: A review. *IEEE Sensors Journal* 22, 22 (2022), 21362–21390.

[29] Soroush Heydari and Qusay H Mahmoud. 2025. Tiny machine learning and on-device inference: A survey of applications, challenges, and future directions. *Sensors* 25, 10 (2025), 3191.

[30] Ismail Lamaakal, Siham Essahraui, Yassine Maleh, Khalid El Makkaoui, Ibrahim Ouahbi, Mouncef Filali Bouami, Ahmed A Abd El-Latif, May Almousa, Jialiang Peng, and Dusit Niyato. 2025. A Comprehensive Survey on Tiny Machine Learning for Human Behavior Analysis. *IEEE Internet of Things Journal* (2025).

[31] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597* (2021).
[32] Yipeng Sun and Andreas M Kist. 2021. Deep learning on edge tpus. *arXiv preprint arXiv:2108.13732* (2021).
[33] Himax Technologies Inc. 2020. Himax Launches WiseEye WE-I Plus HX6537-A to Support AI Deep Learning with Google's TensorFlow Lite for Microcontrollers. (June 2020). https://www.globenewswire.com/news-release/2020/06/30/2055240/8267/en/Himax-Launches-WiseEye-WE-I-Plus-HX6537-A-to-Support-AI-Deep-Learning-with-Google-s-TensorFlow-Lite-for-Microcontrollers.html
[34] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597* (2021).
[35] Peter Torelli and Mohit Bangale. 2021. Measuring inference performance of machine-learning frameworks on edge-class devices with the mlmark benchmark. *Techincal Report. Available online: https://www. eembc. org/techlit/articles/MLMARK-WHITEPAPERFINAL-1. pdf (accessed on 5 April 2021)* (2021).
[36] Colby R Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, et al. 2020. Benchmarking tinyml systems: Challenges and direction. *arXiv preprint arXiv:2003.04821* (2020).
[37] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2016. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530* (2016).
[38] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. 2019. Visual wake words dataset. *arXiv preprint arXiv:1906.05721* (2019).
[39] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. 2020. Mcunet: Tiny deep learning on iot devices. *Advances in neural information processing systems* 33 (2020), 11711–11722.
[40] Sangwon Lee, Jonghoon Choi, Sehoon Park, and Sungroh Yoon. 2020. Designing Extremely Memory-Efficient CNNs for On-Device Vision Tasks. *IEEE Access* 8 (2020), 49401–49413.
[41] Gaurav Menghani. 2023. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *Comput. Surveys* 55, 12 (2023), 1–37.
[42] Gabriel Signoretti, Marianne Silva, Pedro Andrade, Ivanovitch Silva, Emiliano Sisinni, and Paolo Ferrari. 2021. An evolving tinyml compression algorithm for iot environments based on data eccentricity. *Sensors* 21, 12 (2021), 4153.
[43] Urmish Thakker, Paul N Whatmough, Zhi-Gang Liu, Matthew Mattina, and Jesse Beu. 2020. Compressing language models using doped kronecker products. *arXiv preprint arXiv:2001.08896* (2020).
[44] Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. 2020. Starfish: Resilient image compression for AIoT cameras. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 395–408.
[45] Sek M Chai. 2021. Quantization-guided training for compact TinyML models. In *Research Symposium on Tiny Machine Learning*.
[46] Manuele Rusci, Marco Fariselli, Alessandro Capotondi, and Luca Benini. 2020. Leveraging automated mixed-low-precision quantization for tiny edge microcontrollers. In *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning: Second International Workshop, IoT Streams 2020, and First International Workshop, ITEM 2020, Co-located with ECML/PKDD 2020, Ghent, Belgium, September 14-18, 2020, Revised Selected Papers 2*. Springer, 296–308.
[47] Hamed Fatemi, Vedant Karia, Tej Pandit, and Dhireesha Kudithipudi. 2020. TENT: Efficient Quantization of Neural Networks on the Tiny Edge with Tapered Fixed Point. In *Proceedings of the Research Symposium on Tiny Machine Learning*.
[48] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).
[49] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[50] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), 4510–4520.
[51] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
[52] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).
[53] Haokui Zhang, Wenze Hu, and Xiaoyu Wang. 2022. Parc-net: Position aware circular convolution with merits from convnets and transformer. In *European conference on computer vision*. Springer, 613–630.
[54] Yvan Tortorella, Luca Bertaccini, Luca Benini, Davide Rossi, and Francesco Conti. 2023. RedMule: A mixed-precision matrix–matrix operation engine for flexible and energy-efficient on-chip linear algebra and TinyML training acceleration. *Future Generation Computer Systems* 149 (2023), 122–135.
[55] Hasib-Al Rashid, Argho Sarkar, Aryya Gangopadhyay, Maryam Rahnemoonfar, and Tinoosh Mohsenin. 2024. TinyVQA: Compact Multimodal Deep Neural Network for Visual Question Answering on Resource-Constrained Devices. *arXiv preprint arXiv:2404.03574* (2024).

[56] Soroush Heydari and Qusay H Mahmoud. 2025. Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions. *Sensors* 25, 10 (2025), 3191.
[57] Praneel Chand and Mansour Assaf. 2024. An Empirical Study on Lightweight CNN Models for Efficient Classification of Used Electronic Parts. *Sustainability* 16, 17 (2024), 7607.
[58] Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, and Alexander Wong. 2017. Squishednets: Squishing squeezenet further for edge device scenarios via deep evolutionary synthesis. *arXiv preprint arXiv:1711.07459* (2017).
[59] Yuanyuan Xu, Genke Yang, Jiliang Luo, and Jianan He. 2020. An electronic component recognition algorithm based on deep learning with a faster SqueezeNet. *Mathematical Problems in Engineering* 2020, 1 (2020), 2940286.
[60] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of machine learning and systems* 3 (2021), 517–532.
[61] Victor JB Jung, Alessio Burrello, Moritz Scherer, Francesco Conti, and Luca Benini. 2024. Optimizing the deployment of tiny transformers on low-power mcus. *IEEE Trans. Comput.* (2024).
[62] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461 (2021), 370–403.
[63] Benjamin Hawks, Javier Duarte, Nicholas J Fraser, Alessandro Pappalardo, Nhan Tran, and Yaman Umuroglu. 2021. Ps and qs: Quantization-aware pruning for efficient low latency neural network inference. *Frontiers in Artificial Intelligence* 4 (2021), 676564.
[64] Andrey Kuzmin, Markus Nagel, Mart Van Baalen, Arash Behboodi, and Tijmen Blankevoort. 2023. Pruning vs quantization: Which is better? *Advances in neural information processing systems* 36 (2023), 62414–62427.
[65] KA Kumari, S Ahamad, T Patil, K Sardana, E Muniyandy, and D Pilli. 2024. Neural Network Pruning Techniques for Efficient Model Compression. *International Journal of Intelligent Systems and Applications in Engineering* 12, 15s (2024), 565–575.
[66] Han Cai, Chuang Gan, Ji Lin, and Song Han. 2021. Network augmentation for tiny deep learning. *arXiv preprint arXiv:2110.08890* (2021).
[67] Hoang-The Pham, Minh-Anh Nguyen, and Chi-Chia Sun. 2019. AIoT solution survey and comparison in machine learning on low-cost microcontroller. In *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*. IEEE, 1–2.
[68] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems* 3 (2021), 800–811.
[69] Shawn Hymel, Jan Trivedi, Louis Heller, Sandeep Sharma, Paul Fiedler, Ajay Patel, Anil Chandak, Abhishek Sinha, and Thomas Schmid. 2022. Edge Impulse: An MLOps Platform for Tiny Machine Learning. *arXiv preprint* arXiv:2212.03332 (2022). Available at https://arxiv.org/abs/2212.03332.
[70] Google AI Edge. 2025. TensorFlow Lite Model Maker. https://ai.google.dev/edge/litert/libraries/modify. (2025). Accessed: June 5, 2025.
[71] Towards Data Science. 2025. Pytorch — ExecuTorch Documentation. https://docs.pytorch.org/executorch/stable/index.html. (2025).
[72] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *arXiv preprint* arXiv:1801.06601 (2018). Available at https://arxiv.org/abs/1801.06601.
[73] C. Liu, M. Jobst, L. Guo, X. Shi, J. Partzsch, and C. Mayr. 2023. Deploying Machine Learning Models to Ahead-of-Time Runtime on Edge Using MicroTVM. *arXiv preprint* arXiv:2304.04842 (2023). Available at https://arxiv.org/abs/2304.04842.
[74] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein, J. Montgomery, B. Maher, S. Nadathur, J. Olesen, J. Park, A. Rakhov, M. Smelyanskiy, and M. Wang. 2018. Glow: Graph Lowering Compiler Techniques for Neural Networks. *arXiv preprint* arXiv:1805.00907 (2018). Available at https://arxiv.org/abs/1805.00907.
[75] 2021. Keras2c: A library for converting Keras neural networks to real-time compatible C. *Engineering Applications of Artificial Intelligence* 100 (2021), 104188. DOI: http://dx.doi.org/10.1016/j.engappai.2021.104188
[76] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray. 2013. MLPACK: A Scalable C++ Machine Learning Library. *Journal of Machine Learning Research* 14, 1 (2013), 801–805.
[77] STMicroelectronics. 2025. X-CUBE-AI: STM32Cube Expansion Package. https://www.st.com/resource/en/data_brief/x-cube-ai.pdf. (2025). Accessed: June 5, 2025.
[78] J. Duarte, E. Kreinar, J. Ngadiuba, et al. 2021. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. In *TinyML Research Symposium 2021, San Jose, CA*. arXiv:2103.05579, https://arxiv.org/abs/2103.05579.
[79] ARM Ltd. 2025. OctoML: Accelerating ML model deployment. https://www.arm.com/partners/catalog/octoml. (2025). ARM Partner Catalog, Accessed: June 5, 2025.
[80] Nebuly Team. 2024. nebullvm: AI runtime optimization library. https://pypi.org/project/nebullvm/. (2024). Accessed: June 5, 2025.
[81] Mauro Conti, Roberto Di Pietro, Luigi V. Mancini, and Alessandro Mei. 2009. (old) Distributed data source verification in wireless sensor networks. *Inf. Fusion* 10, 4 (2009), 342–353. DOI: http://dx.doi.org/10.1016/j.inffus.2009.01.002
[82] X. He. 2023. Accelerated linear algebra compiler for computationally efficient numerical models. *PLOS ONE* 18, 2 (2023), e0282265. DOI: http://dx.doi.org/10.1371/journal.pone.0282265
[83] J. Bai, F. Lu, K. Zhang, et al. 2019. ONNX: Open Neural Network Exchange. GitHub repository. (2019). https://github.com/onnx/onnx.

[84] N. Vasilache et al. 2018. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions. *arXiv preprint* arXiv:1802.04730 (2018). Available at https://arxiv.org/abs/1802.04730.
[85] Neuton.AI. Neuton AI User Guide. https://neuton.ai/uploads/user-guide.pdf. ([n. d.]). Accessed: November 6, 2025.
[86] LatentAI. 2025. *From Cloud-First to Edge-First: The Future of Enterprise AI.* White Paper. LatentAI. https://latentai.com/wp-content/uploads/2025/05/Cloud-to-Edge-White-Paper-FINAL.pdf.
[87] Arduino. 2025. Get Started with Machine Learning on Arduino Nano 33 BLE Sense. https://docs.arduino.cc/tutorials/nano-33-ble-sense/get-started-with-machine-learning/. (2025). Accessed: 2025-06-11.
[88] NXP Semiconductors. 2024. eIQ Toolkit User Guide. https://www.nxp.com/docs/en/user-guide/EIQTKUG-1.8.0.pdf. (2024). Version 1.8.0.
[89] Dennis, Don Kurian and Gopinath, Sridhar and Gupta, Chirag and Kumar, Ashish and Kusupati, Aditya and Patil, Shishir G and Simhadri, Harsha Vardhan. EdgeML: Machine Learning for resource-constrained edge devices. ([n. d.]). https://github.com/Microsoft/EdgeML.
[90] EdjeElectronics. 2025. TensorFlow Lite Object Detection on Android and Raspberry Pi. https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi. (2025). Accessed: 2025-06-11.
[91] Sony. 2025. Model Optimization Toolkit. https://github.com/sony/model_optimization. (2025). Accessed: 2025-06-11.
[92] Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. 2017. ProtoNN: compressed and accurate kNN for resource-scarce devices. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. JMLR.org, 1331–1340.
[93] Yuxuan Liang, Yulin Han, and Fangming Jiang. 2022. Deep Learning-based Small Object Detection: A Survey. In *Proceedings of the 2022 8th International Conference on Computing and Artificial Intelligence (ICCAI '22)*. ACM, 432–438. DOI: http://dx.doi.org/10.1145/3530249.3530294
[94] Qianyun Lu and Boris Murmann. 2022. Improving the Energy Efficiency and Robustness of tinyML Computer Vision using Log-Gradient Input Images. In *Proceedings of the tinyML Research Symposium (tinyML Research Symposium '22)*. ACM. Also available as arXiv:2203.02571.
[95] Colby Banbury, Emil Njor, Andrea Mattia Garavagno, Mark Mazumder, Matthew Stewart, Pete Warden, Manjunath Kudlur, Nat Jeffries, and Vijay Janapa Reddi. 2025. Wake Vision: A Tailored Dataset and Benchmark Suite for TinyML Computer Vision Applications. *arXiv preprint arXiv:2405.00892v5* (Jun 2025). DOI: http://dx.doi.org/10.48550/arXiv.2405.00892 ver.5.
[96] S. You, Zhiyu Chen, Shangdong Li, Mengxue Wang, Tengfeng Feng, and Yimu Jiang. 2022. YoLite+: a Lightweight Multi-Object Detection Approach in Traffic Scenarios. *Procedia Computer Science* 199 (2022), 346–353.
[97] Andrew Barovic and Armin Moin. 2025. TinyML for Speech Recognition. *arXiv preprint arXiv:2504.16213* (2025).
[98] Ahmed Y. Radwan, Mohammad Shehab, and Mohamed-Slim Alouini. 2025. TinyML NLP Scheme for Semantic Wireless Sentiment Classification with Privacy Preservation. *arXiv preprint arXiv:2411.06291v3* (April 2025). https://arxiv.org/abs/2411.06291 Accepted at EuCNC & 6G Summit 2025.
[99] Ismail Lamaakal, Yassine Maleh, Khalid El Makkaoui, Ibrahim Ouahbi, Mohamed Essahraui, Mohamed F. Bouami, Ahmed A. Abd El-Latif, May Almousa, Jun Peng, and Dusit Niyato. 2025. A Comprehensive Survey on Tiny Machine Learning for Human Behavior Analysis (HBA). *IEEE Internet of Things Journal* (2025). https://ieeexplore.ieee.org/document/12345678 In press.
[100] Mohammad Wali Ur Rahman, Murad Mehrab Abrar, Hunter Gibbons Copening, Salim Hariri, Sicong Shao, Pratik Satam, and Soheil Salehi. 2023. Quantized Transformer Language Model Implementations on Edge Devices. In *Proceedings of the 2023 IEEE 22nd International Conference on Machine Learning and Applications (ICMLA)*. 104–111. DOI: http://dx.doi.org/10.1109/ICMLA58977.2023.00104
[101] Zhaolan Huang, Adrien Tousnakhoff, Polina Kozyr, Roman Rehausen, Felix Bießmann, Robert Lachlan, Cedric Adjih, and Emmanuel Baccelli. 2024. TinyChirp: Bird Song Recognition Using TinyML Models on Low-power Wireless Acoustic Sensors. *arXiv preprint arXiv:2407.21453v2* (Sept. 2024). https://arxiv.org/abs/2407.21453v2 Accepted at IEEE IS2 2024.
[102] Norhen Abdennadher, Danilo Pau, and Arcangelo Bruna. 2021. Fixed complexity tiny reservoir heterogeneous network for on-line ECG learning of anomalies. In *2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*. IEEE, 233–237.
[103] Anandi Dutta. 2016. *A smart design framework for a novel reconfigurable multi-processor systems-on-chip (ASREM) architecture.* Ph.D. dissertation. University of Louisiana at Lafayette, Lafayette, LA, USA.
[104] Martin Ragot, Nicolas Martin, Sonia Em, Nico Pallamin, and Jean-Marc Diverrez. 2018. Emotion recognition using physiological signals: laboratory vs. wearable sensors. In *Advances in Human Factors in Wearable Technologies and Game Design: Proceedings of the AHFE 2017 International Conference on Advances in Human Factors and Wearable Technologies, July 17-21, 2017, The Westin Bonaventure Hotel, Los Angeles, California, USA 8*. Springer, 15–22.
[105] Satyapreet Singh Yadav, Radha Agarwal, Kola Bharath, Sandeep Rao, and Chetan Singh Thakur. 2022. TinyRadar: MmWave radar based human activity classification for edge computing. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2414–2417.
[106] Bidyut Saha, Riya Samanta, Soumya Kanti Ghosh, and Ram Babu Roy. 2023. From Wrist to World: Harnessing Wearable IMU Sensors and TinyML to Enable Smart Environment Interactions. In *Proceedings of the Third International Conference on AI-ML Systems*. 1–3.
[107] J Manokaran and G Vairavel. 2022. Smart anomaly detection using data-driven techniques in iot edge: a survey. In *Proceedings of Third International Conference on Communication, Computing and Electronics Systems: ICCCES 2021*. Springer, 685–702.

[108] Vítor M Oliveira and António HJ Moreira. 2021. Edge AI system using a thermal camera for industrial anomaly detection. In *International Summit Smart City 360°*. Springer, 172–187.
[109] Matteo Cardoni, Danilo Pietro Pau, Laura Falaschetti, Claudio Turchetti, and Marco Lattuada. 2021. Online learning of oil leak anomalies in wind turbines with block-based binary reservoir. *Electronics* 10, 22 (2021), 2836.
[110] Nikesh Gondchawar, RS Kawitkar, et al. 2016. IoT based smart agriculture. *International Journal of advanced research in Computer and Communication Engineering* 5, 6 (2016), 838–842.
[111] Devis Tuia, Benjamin Kellenberger, Sara Beery, Blair R Costelloe, Silvia Zuffi, Benjamin Risse, Alexander Mathis, Mackenzie W Mathis, Frank Van Langevelde, Tilo Burghardt, et al. 2022. Perspectives in machine learning for wildlife conservation. *Nature communications* 13, 1 (2022), 792.
[112] Kong Ka Hing, Mehran Behjati, Vala Saleh, Yap Kian Meng, Anwar PP Abdul Majeed, and Yufan Zheng. 2024. Edge Intelligence for Wildlife Conservation: Real-Time Hornbill Call Classification Using TinyML. In *International Conference on Intelligent Manufacturing and Robotics*. Springer, 476–488.
[113] Mattia Antonini, Miguel Pincheira, Massimo Vecchio, and Fabio Antonelli. 2022. A TinyML approach to non-repudiable anomaly detection in extreme industrial environments. In *2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4. 0&IoT)*. IEEE, 397–402.
[114] Muhammad Abubakar, Adbul Sattar, Hamid Manzoor, Khola Farooq, and Muhammad Yousif. 2025. IIOT: An Infusion of Embedded Systems, TinyML, and Federated Learning in Industrial IoT. *Journal of Computing & Biomedical Informatics* 8, 02 (2025).
[115] Lachit Dutta and Swapna Bharali. 2021. Tinyml meets iot: A comprehensive survey. *Internet of Things* 16 (2021), 100461.
[116] Mangesh Pujari, Anil Kumar Pakina, and Ashwin Sharma. 2023. Enhancing cybersecurity in edge AI systems: A game-theoretic approach to threat detection and mitigation. *IOSR Journal of Computer Engineering* 25, 3 (2023), 65–73.
[117] Haoyu Ren, Darko Anicic, and Thomas A Runkler. 2021. Tinyol: Tinyml with online-learning on microcontrollers. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.
[118] Mark Mazumder, Colby Banbury, Josh Meyer, Pete Warden, and Vijay Janapa Reddi. 2021. Few-shot keyword spotting in any language. *arXiv preprint arXiv:2104.01454* (2021).
[119] Kavya Kopparapu and Eric Lin. 2021. TinyFedTL: Federated transfer learning on tiny devices. *arXiv preprint arXiv:2110.01107* (2021).
[120] Marc Monfort Grau, Roger Pueyo Centelles, and Felix Freitag. 2021. On-device training of machine learning models on microcontrollers with a look at federated learning. In *Proceedings of the Conference on Information Technology for Social Good*. 198–203.
[121] M. Pujari, A. Goel, and A. K. Pakina. 2024. Efficient TinyML Architectures for On-Device Small Language Models: Privacy-Preserving Inference at the Edge. *International Journal Science and Technology* 3, 3 (2024), 67–75.
[122] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[123] Pete Warden and Daniel Situnayake. 2019. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media.
[124] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.
[125] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. 2016. Memory-efficient backpropagation through time. *Advances in neural information processing systems* 29 (2016).
[126] Minsik Cho and Daniel Brand. 2017. MEC: Memory-efficient convolution for deep neural network. In *International Conference on Machine Learning*. PMLR, 815–824.
[127] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 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*. 2704–2713.
[128] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).
[129] Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems* 32 (2019).
[130] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 8612–8620.
[131] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).
[132] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned step size quantization. *arXiv preprint arXiv:1902.08153* (2019).
[133] TensorFlow Model Optimization Toolkit. https://www.tensorflow.org/model_optimization. ([n. d.]). Accessed: 2024-06-10.

[134] Boyu Chen, Peixia Li, Baopu Li, Chen Lin, Chuming Li, Ming Sun, Junjie Yan, and Wanli Ouyang. 2021. Bn-nas: Neural architecture search with batch normalization. In *Proceedings of the IEEE/CVF international conference on computer vision*. 307–316.
[135] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019).
[136] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. 2019. Pulp-nn: A computing library for quantized neural network inference at the edge on risc-v based parallel ultra low power clusters. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 33–36.
[137] Ambiq. 2023. *Choosing the Best Memory for Developer AI Model White Paper*. White Paper. Ambiq. https://ambiq.com/wp-content/uploads/2023/06/Choosing-the-Best-Memory-for-Developer-AI-Model-WP.pdf.
[138] J Lin, WM Chen, H Cai, C Gan, and S Han. 2021. MCUNetV2: Memory-Efficient Patch-based Inference for Tiny Deep Learning. arXiv. *arXiv preprint arXiv:2110.15352* (2021).
[139] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).
[140] Yann Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.
[141] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report. University of Toronto. https://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf
[142] Riya Adlakha and Eltahir Kabbar. 2024. The challenges of TinyML implementation: A literature review. In *Proceedings: CITRENZ 2023 Conference, Auckland, 27–29 September*, Hessam Sharifzadeh (Ed.). Unitec ePress, Te Pūkenga - New Zealand Institute of Skills and Technology, Auckland, New Zealand, 160–169. DOI: http://dx.doi.org/10.34074/proc.240120
[143] Filip Svoboda, Javier Fernandez-Marques, Edgar Liberis, and Nicholas D Lane. 2022. Deep learning on microcontrollers: A study on deployment costs and challenges. In *Proceedings of the 2nd European Workshop on Machine Learning and Systems*. 54–63.
[144] Parin Shah, Yuvaraj Govindarajulu, Pavan Kulkarni, and Manojkumar Parmar. 2024. Enhancing TinyML Security: Study of Adversarial Attack Transferability. *arXiv preprint arXiv:2407.11599* (2024).
[145] Jacob Huckelberry, Yuke Zhang, Allison Sansone, James Mickens, Peter A Beerel, and Vijay Janapa Reddi. 2024. TinyML Security: Exploring Vulnerabilities in Resource-Constrained Machine Learning Systems. *arXiv preprint arXiv:2411.07114* (2024).
[146] Archit Parnami and Minwoo Lee. 2022. Learning from few examples: A summary of approaches to few-shot learning. *arXiv preprint arXiv:2203.04291* (2022).
[147] Yeonju Kim, Jeonghyeon Yoon, and Seungku Kim. 2025. A Few-Shot Learning-Based Material Recognition Scheme Using Smartphones. *Applied Sciences* 15, 1 (2025), 430.
[148] Enrico Fini, Stéphane Lathuiliere, Enver Sangineto, Moin Nabi, and Elisa Ricci. 2020. Online continual learning under extreme memory constraints. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII 16*. Springer, 720–735.
[149] Sukhpal Singh Gill, Muhammed Golec, Jianmin Hu, Minxian Xu, Junhui Du, Huaming Wu, Guneet Kaur Walia, Subramaniam Subramanian Murugesan, Babar Ali, Mohit Kumar, et al. 2025. Edge AI: A taxonomy, systematic review and future directions. *Cluster Computing* 28, 1 (2025), 1–53.
[150] Xubin Wang, Zhiqing Tang, Jianxiong Guo, Tianhui Meng, Chenhao Wang, Tian Wang, and Weijia Jia. 2025. Empowering Edge Intelligence: A Comprehensive Survey on On-Device AI Models. *Comput. Surveys* 57, 9 (2025), 1–39.

Received 23 June 2025; revised 24 September 2025; accepted 21 October 2025