VOL. 24, NO. 2, 2025, 81-97
elektrika.utm.my
ISSN 0128-4428

ELEKTRIKA
Journal of Electrical Engineering

A Comprehensive Review of Tiny Machine Learning: Enabling AI on Resource-Constrained Devices

Nurul Atikah Abbas[1] and **Mohd Ridzuan Ahmad**[1*]

[1]Division of Control and Mechatronics Engineering, School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia

[*]Corresponding author: mdridzuan@utm.my

Abstract: Recently, there has been a growing focus on artificial intelligence (AI), particularly on a specific type known as machine learning. This technology is extensively utilized in smart devices. However, traditional machine learning methods require large models and substantial computing power to operate effectively. In response to this challenge, the concept of "tiny machine learning" (TinyML) emerged. TinyML aims to enhance machine learning capabilities on small and resource-constrained devices by enabling them to perform actions locally, without relying on cloud-based resources. TinyML represents a relatively new and exciting field that combines machine learning with embedded devices, which are often compact and energy-efficient. The primary goal is to ensure that even when transitioning from high-end systems to low-end devices, the performance and accuracy of machine learning are well-maintained. The focus of this paper is to provide an overview of the current state-of-the-art in TinyML, covering various aspects such as background information, software implementations, hardware requirements, and the frameworks utilized in this emerging field.

Keywords: TinyML, artificial intelligence, edge AI, optimization, embedded device

Article History: received 5 March 2024; accepted 22 May 2025; published 31 August 2025

1. INTRODUCTION

The Internet of Things (IoT) refers to a network of interconnected computing devices that can gather, transmit, and respond to data collected from various sensors [1]. IoT is a method of linking common objects or "things" to the internet so they may interact, communicate, and be monitored or controlled remotely.

There are six IoT essential elements: (i) Device: These are the actual things or gadgets that include sensors, actuators, and communication capabilities. Examples include wearable technology, smart home appliances, industrial machinery, smart thermostats, and smart home wearables; (ii) Connectivity: IoT devices connect to the internet and interact with one another via a variety of communication protocols and technologies, including Wi-Fi, Bluetooth, Zigbee, cellular networks, and low-power wide-area networks (LPWAN); (iii) Data processing: IoT devices use their sensors to produce and collect data, which is then either processed locally on the device or on the cloud; (iv) Cloud computing: IoT devices frequently communicate the data they gather to cloud-based servers for larger-scale archiving, processing, and analysis. The processing and storing of enormous volumes of data from various IoT devices is made possible via cloud computing; (v) Data analytics: The data gathered from IoT devices is analyzed to uncover important insights, trends, and patterns that may be used for decision-making and process improvement; (vi) Actions and Automation: IoT systems may initiate actions or automate procedures based on the insights discovered through data analysis, enabling real-time answers and improved efficiency.

Smart homes, healthcare, agriculture, transportation, industrial automation, and other fields have all benefited from the use of IoT. It promises to fundamentally alter how we engage with technology and how machines and systems coordinate to improve the convenience and effectiveness of our daily lives. The IoT ecosystem has significant issues in guaranteeing security, privacy, and data protection as the number of connected devices increases.

To meet the demands of evolving technology, embedded systems were introduced, capable of sensing, communicating, and generating outputs with low power consumption, small size, and cost-effectiveness [2]. In IoT systems, microcontrollers are frequently employed as edge devices due to their limited processing capacity and memory footprint [3][4]. These microcontrollers typically consume very little power (in the order of microwatts) and have a few hundred kilobytes of flash memory [5]. However, their limited resources have made them less suitable for employing machine learning models [6]. Nonetheless, edge devices offer advantages in terms of their energy efficiency and reasonably acceptable accuracy. One of the significant challenges faced by traditional IoT systems is the need to process data on the cloud, which introduces latency to the system. This delay

in data processing can prevent real-time responses and cause inefficiencies in certain applications.

Transformative improvements in intelligent technology are now possible because of the effortless combination of artificial intelligence (AI) and the Internet of Things (IoT). AI algorithms can handle and analyze enormous volumes of data by utilizing the data-rich environment provided by IoT devices, providing real-time insights, autonomous decision-making, and predictive capabilities. The integrated ecosystem of IoT and AI-driven data enables businesses to optimize operations, improve customer experiences, and provide individualized offerings. The potential for game-changing breakthroughs and increased efficiency in several industries is becoming more and more obvious as the synergy between AI and IoT continues to emerge. To enable the appropriate and advantageous deployment of AI-driven IoT applications, thorough consideration of privacy, security, and ethical issues is necessary.

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think, learn, and perform tasks that typically require human intelligence. The primary goal of AI is to create systems that can imitate human cognitive abilities, such as problem-solving, reasoning, learning, perception, understanding natural language, and decision-making.

AI could be divided into two main types which are Narrow AI and artificial general intelligence (AGI). The difference between them is based on how well they can solve problems. Narrow AI is designed to do specific tasks or solve particular problems. It works really well in one area and can even perform as well as humans in that field. Examples include voice assistants like Siri and Alexa, recommendation systems on Netflix or Spotify, and computer programs that are really good at games like chess. AGI, on the other hand, is a type of AI that doesn't exist yet. It's a future goal — a system that would have human-like intelligence and could understand, learn, and apply knowledge to a wide variety of tasks, just like we do. This kind of AI would be truly self-aware and capable of independent thought, but for now, it's still something we're working toward.

Machine learning, natural language processing (NLP), computer vision, robotics, and expert systems are just a few of the methods and techniques that AI uses. A key factor in the advancement of AI has been machine learning, which enables systems to learn from data and enhance their performance over time without being explicitly programmed for each task.

Artificial Intelligence (AI) is a specialized branch of computer science that encompasses the creation and utilization of machine learning (ML) techniques to process and extract valuable information from raw data [7]. ML models, being complex, demand significant computational power and memory to execute and analyze effectively. Consequently, they are best suited for high-performance computing systems [6]. Conventional machine learning typically relies on cloud-based systems, necessitating internet connectivity for data processing [8].

However, this reliance on cloud-based processing introduces several drawbacks, such as increased latency, privacy concerns, and higher energy consumption [9]. Despite these challenges, machine learning has shown impressive results in various applications, often surpassing human-level classification accuracy [10]. As a result, ML finds application in diverse fields, including human-computer interaction, agriculture, autonomous vehicles, assistive devices, healthcare, smart cities, and monitoring systems. As AI technology develops, it has the potential to revolutionize a number of sectors, increase productivity, and improve people's quality of life in general. Its versatility and potential for significant impact continue to drive research and development in various domains. However, ethical issues, openness, and responsible AI use continue to be crucial aspects of its research and use.

The current advancements in both the Internet of Things (IoT) and Artificial Intelligence (AI) have opened up new possibilities for extending Edge Machine Learning (ML) capabilities using microcontrollers, leading to the emergence of TinyML [11]. This combination of technologies has garnered significant attention from both the academic and industrial communities [12].

Traditional IoT and AI systems often rely on central cloud management, which can give rise to concerns regarding privacy, reliability, and latency. The primary objective of TinyML is to implement machine learning directly on low-end devices, enabling them to make decisions autonomously without the need to constantly stream data to the cloud [5]. A key focus of TinyML is to compress machine learning models to fit the requirements of the target device. Implementing machine learning inference on embedded devices offers several advantages:

i. Improved Latency: By eliminating the need for communication with cloud servers, the processing time is significantly reduced, leading to lower latency.
ii. Enhanced Privacy and Security: As data remains on the device itself, there is a higher level of privacy and security, reducing the risk of sensitive information being exposed to external parties.
iii. Increased Energy Efficiency: By excluding the need for constant computation and communication with the cloud, energy consumption is minimized, resulting in improved energy efficiency [2].

Overall, TinyML represents a promising approach to bringing machine learning capabilities to resource-constrained devices while addressing the limitations associated with traditional IoT and AI systems. However, despite its numerous advantages, TinyML is not without its challenges. Addressing the limitations related to low-end embedded devices is crucial to unleashing the full potential of this technology. Therefore, we will examine the obstacles in implementing TinyML, offering insights into the strategies employed to overcome them.

In this research paper, the focus is on exploring the evolution of TinyML, which has progressed from using very large machine learning models to more optimized versions. The paper delves into various aspects of TinyML, including its tools and state-of-the-art applications. Additionally, it discusses the industrial potential of TinyML and highlights the challenges that come with

implementing machine learning on low-end embedded devices. The major contributions of the paper are as follows:

i. Providing an Overview of TinyML: The paper presents a comprehensive overview of TinyML, including its background, to help readers understand the fundamentals and workings of this technology.
ii. Illustrating Existing TinyML Infrastructure: The research paper explores the embedded devices used in TinyML, along with the software, frameworks, and datasets employed in current TinyML implementations.
iii. Presenting State-of-the-Art Frameworks: The paper discusses and showcases the latest and most advanced frameworks used in TinyML applications.
iv. Analyzing Use Cases for TinyML: The research paper examines various use cases where TinyML is applied, focusing on the performance and energy efficiency of the system.
v. Discussing Challenges with Low-End Devices: The paper addresses the challenges that arise when utilizing low-end embedded devices for TinyML and offers insights into potential solutions.

By covering these key aspects, the paper contributes to a better understanding of TinyML, its potential applications, and the hurdles that need to be overcome to unlock its full capabilities on resource-constrained devices.

2. BACKGROUND OF TINYML

In recent years, the convergence of cutting-edge technologies, such as the Internet of Things (IoT) and Artificial Intelligence (AI), has revolutionized the landscape of computing and data processing. As a result, the concept of Tiny Machine Learning (TinyML) has emerged, empowering resource-constrained devices to perform complex tasks through local machine learning models. It entails installing portable, low-power machine learning models right onto compact embedded devices, allowing them to carry out difficult tasks locally without constant cloud access. TinyML creates new opportunities for edge computing and real-time, intelligent applications across a variety of fields by compressing the machine learning algorithms and optimizing them for deployment on devices with constrained CPU and memory resources. This novel paradigm has sparked significant interest in both the academic and industrial spheres, paving the way for unprecedented advancements in edge computing.

Tiny Machine Learning has developed as a result of improvements in both hardware and software. At first, machine learning models were too complex and computationally demanding to function well on entry-level hardware. However, TinyML became a practical reality when hardware capabilities increased, such as the availability of energy-efficient microcontrollers and specialized hardware accelerators[13]. Transfer learning has emerged as a vital enabler of Tiny Machine Learning (TinyML), offering a practical solution for developing effective models in environments with limited computational resources and small datasets. By reusing knowledge from pre-trained models, transfer learning allows for improved model performance on new, related tasks without the need for extensive data collection. In addition, advanced techniques such as model quantization, pruning, and knowledge distillation have been introduced to compress large, complex models into lightweight versions suitable for edge deployment. These developments collectively make TinyML more accessible and practical for real-world applications where memory, power, and data availability are constrained.

The development of TinyML began with traditional machine learning and mobile machine learning. The development of the machine learning system coincides with the expansion of cloud computing in data processing in 2006. Deep neural network research and development are always becoming more sophisticated. Machine learning techniques that are traditional or cloud-based can handle neural networks with millions of parameters quickly and efficiently. The major emphasis of traditional machine learning, which is accuracy, is a sizable model of neural networks. This is because memory and computing capacity are unbounded.

Mobile computing has become more popular after a decade of cloud-based machine learning technologies and an increase in mobile device usage. Mobile computing is essentially communication with a limited resource via a mobile device. The neural network technique cannot execute on the devices due to the low-power CPU and little memory [14]. To create a light neural network, optimization is therefore required. This real-time computation for mobile or edge machine learning is needed, independent of cloud computing resources. New types of neural networks, which are smaller in size and concentrated on the model's correctness and efficiency, have been developed as a result of the transition to mobile platforms [15]. Many industries, including robotics and augmented reality, have employed mobile machine learning.

With the development of artificial intelligence and the internet of things, the microcontroller will experience exponential growth in the next years. There are 250 billion microcontrollers in use today, and 40 billion are expected to be in use over the next two years [16]. One of the embedded devices is a microcontroller, which gathers data from sensors and sends it to the cloud for processing. On embedded devices, sensors like accelerometers, cameras, microphones, and temperature sensors are frequently employed. This application will result in delays, data loss, and privacy problems. The majority of embedded devices have low power ranges and other resource limitations. To operate the neural networks on the device, the prerequisites must be met. New machine learning and neural networks are being developed that are smaller in size, and more focused on lowering latency and privacy while yet being able to retain accuracy.

The evolution of TinyML is shown in Figure 1. TinyML is a new approach and an emerging field that at the intersection of ML and embedded systems brings the performative power of machine learning to shrink deep structures earning networks to fit on tiny hardware as illustrated in Figure 2. Specifically, TinyML only focuses

on the development and deployment of ML models on low-power devices and involves 3 elements which are software, hardware, and algorithms. Therefore, the ML model is optimized to run on resource-constrained devices like microcontrollers. By enabling the low-power device to run ML, the inferencing process will be in real-time on the device itself providing great responsiveness and privacy [17]. A low-power device usually has a limited size of input and number of layers due to small memory which is only a few hundred kB of memory [18]. TinyML systems should be optimized to provide better accuracy. Further study is needed to develop co-design strategies that tightly integrate hardware and algorithmic design to maximize performance and energy savings in real-world deployments.

These days, machine learning technology is cloud-based. But a cloud-based system, which relies on the internet, comes with certain drawbacks, including (i) latency concerns, (ii) privacy issues, (iii) excessive energy consumption, and (iv) being internet-dependent. As a result, TinyML is a separate system that gathers data and processes the results directly on the hardware to enhance performance in terms of latency, energy usage, and privacy. There are several limitations that must be overcome for TinyML to expand. To create a machine learning system on a low-power device, three crucial steps must be taken. First, in terms of energy, embedded devices needed a battery with a minimum capacity of 10 to 100 mAh for stand-alone processing. The majority of gadgets only have clock speeds of 10 to 1000 MHz. Because of this, a sophisticated machine-learning model cannot function effectively on board. One of the significant effect limits is memory shortage since low-end embedded systems often have less than 1MB of onboard flash memory and 512 KB SRAM. This highlights a key research gap: how to systematically evaluate the trade-offs between model compression techniques and real-time performance in various edge scenarios, particularly for safety-critical applications.

To summarise, Tiny Machine Learning represents an exciting and promising frontier in AI and IoT, enabling intelligent and real-time processing on low-power, edge devices. Although it offers many benefits, it also has built-in difficulties that need continual study and innovation to realize its full potential across a range of applications. The creation of effective and reliable TinyML solutions holds the key to improving a number of sectors and altering the way we interact with intelligent technology as the field continues to progress.

3. HARDWARE

We present Table 1 to compare the processor, CPU clock frequency, flash memory, and SRAM capacity of the hardware platforms. We see that the majority of hardware boards operate at processor frequencies below 100 MHz and typically have less than 1 MB of flash and SRAM. Such boards have a power consumption in the mW range. The majority of the gadgets can run on Li-Po and coin batteries in addition to a standard DC power source. We also observe that, among all available options, ARM Cortex-M4 is the most often used CPU.

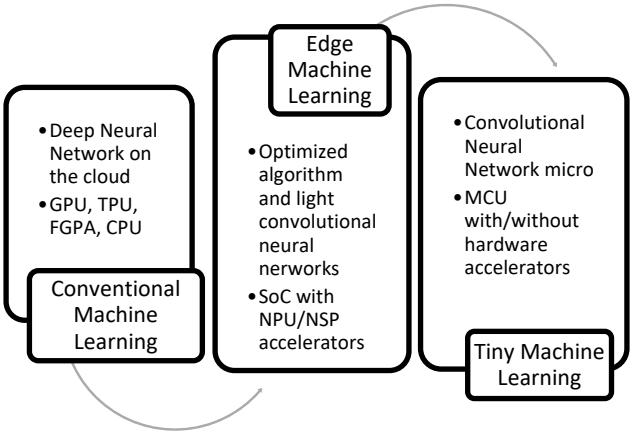

Figure 1. The evolution of TinyML

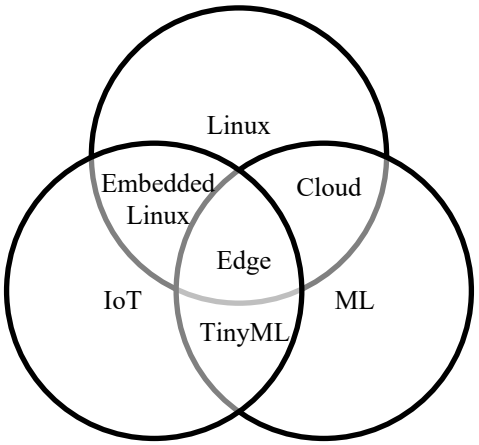

Figure 2. Definition of TinyML

In summary, the hardware landscape for TinyML is rapidly evolving, supporting a wide range of edge AI applications. The growing availability of energy-efficient microcontrollers and specialized processors is making it possible to run meaningful AI workloads directly on small, low-power devices. As hardware continues to improve, TinyML will become even more capable, further expanding its role in real-time, privacy-preserving, and distributed AI systems. Benchmarking the energy and performance trade-offs across many hardware platforms, especially under a range of real-world workloads, is still a significant research gap. Standardized assessment frameworks are also required to direct hardware-software co-design for particular application needs.

Table 1. Hardware

Hardware	Processor	Frequency	SRAM	Flash
Adafruit EdgeBadge [19]	ATSAMD51	120 MHz	192 KB	512 KB
Apollo3 [20]	32-bit ARM	48 MHz	384 KB	1 MB

	Cortex-M4F			
Arduino Nano 33 BLE Sense [21]	nRF52840	64 MHz	256 KB	1 MB
Arduino Portenta H7 [22]	ARM Cortex-M7, ARM Cortex-M4 GPU	480 MHz, 240 MHz	8 MB SDRAM	16 MB
FRDM-K64F [23]	ARM Cortex M4	120MHz	256 KB	1 MB
GAP8 [24]	RISC-V, hardware convolution engine	250 MHz (FC), 175 MHz (C), 22.65GOPs	80 KB,8 MB SDRAM	512 KB
GAP9 [25]	RISC-V, hardware convolution engine	400 MHz, 150.8GOPs	128 KB, 2 MB External	1.5 MB
Nicla Sense ME [26]	ARM Cortex M4	64 MHz	64 KB	512 KB
Nordic Semi nRF52840 DK [27]	ARM Cortex M4	64 MHz	24 KB	192 KB
Nordic Semi Thingy:91 [28]	ARM Cortex M33, nRF9160 SiP	64 MHz	256 KB	1 MB
OpenMV Cam H7 Plus [29]	ARM Cortex-M7	480 MHz	1 MB, 32 MB SDRAM	2 MB (Internal)
Pico4ML BLE [30]	RP2040 DSP dual core	133 MHz	264 KB	4 MB
Raspberry Pi 4B [31]	64-bit ARM Cortex-A72 quad core, Broadcom BCM271	1.5 GHz	256 KB	-
TinyML Board [32]	Syntiant NDP101 NDP, 32-bit ARM Cortex-M0	48 MHz	32 KB	256 KB

4. FRAMEWORK

4.1 TensorFlow Lite

TensorFlow Lite is an open-source deep learning framework for inference on edge devices developed by Google. This framework is special for embedded devices as it contains a converter to convert a TensorFlow model into a compressed version to reduce the size of the model. The flow in using TensorFlow Lite is by picking a model, converting the TF model into a compressed flat buffer (.tflite), loading the .tflite to an embedded edge device, and quantizing the 32-bit floats 8-bit integers [33]. It supports Android, iOS, embedded Linux, and a variety of microcontrollers.

A specific framework called TensorFlow Lite for Microcontrollers is developed with the aim to run machine learning in kB size of memory on 32-bit microcontrollers. The frameworks support a subset of TensorFlow operations. However, it doesn't support model training. Arduino Nano 33 BLE Sense, SparkFun Edge, STM32F746 Discovery kit, Adafruit EdgeBadge, Adafruit TensorFlow Lite for Microcontrollers Kit, Adafruit Circuit Playground Bluefruit, Espressif ESP32-DevKitC, Espressif ESP-EYE, Wio Terminal: ATSAMD51, Himax WE-I Plus EVB Endpoint AI Development Board, Synopsys DesignWare ARC EM Software Development Platform, and Sony Spresense are the development boards supported by TensorFlow Lite.

4.2 Edge Impulse

Edge Impulse is the development platform for machine learning in targeted edge devices. The development of the model starts with data collection and design. Then, training is done on the cloud and the trained model can be deployed on the edge device. The impulse can be directly run on a mobile phone or computer, or it can be exported to a library or build firmware [34]. The developed model does not depend on the internet connection and cloud to run. Therefore, it can reduce the latency and only low power is required.

4.3 Embedded Learning Library (ELL)

ELL is an open-source library for embedded artificial intelligence and machine learning by Microsoft. The aim of ELL is to build a machine learning model that can fit the resource-constrained platforms and provide support for Raspberry Pi, Arduino, and micro:bit [35]. The deployed model works by itself, therefore it doesn't need a network connection.

4.4 Arm-NN

ARM-NN is an open-source Linux software for machine learning run on edge devices. Arm NN SDK makes effective use of the Compute Library to target programmable cores, such as Cortex-A CPUs and Mali GPUs. Cortex-M CPUs are not supported by Arm NN. The most recent version works with ONNX, Caffe, TensorFlow, and TensorFlow Lite. Arm NN deploys networks effectively on Cortex-A CPUs and, if present, Mali GPUs using the Compute Library after taking

networks from various frameworks and translating them to the internal Arm NN format [36].

4.5 PyTorch Mobile

PyTorch, a machine learning framework for a mobile device. It is available for Linux, android, and iOS systems. It also supports GPU, digital signal processor (DSP), and natural processing units (NPU). The advantage of using PyTorch is it supports tracing and scripting via TorchScript IR [37]. For Arm CPUs, there is support for the XNNPACK floating point kernel libraries and QNNPACK integration for 8-bit quantized kernels includes assistance with dynamic quantization, per-channel quantization, and more.

4.6 uTensor

uTensor is an embedded machine learning architecture that is free and open source and intended for quick development and deployment. The project provides a data-collecting framework, a fully configurable graph processing tool, and an inference engine. Only a few kilobytes of RAM, as low as 2kB are required by the device code [38].

4.7 NanoEdge AI Studio

Desktop software for anomaly and outlier identification, feature categorization, and extrapolation of temporal and multivariable data that is specialized for STM32 [39]. Small datasets are used to create anomaly detection libraries: Learn normality on the STM32 microcontroller first-hand and see flaws immediately. One-class classification libraries with very little sample for outlier detection acquisition when the device is in regular operation and the identification of any unusual pattern deviation. Libraries for N-class classification developed on a very small, labeled dataset: real-time signal classification. Extrapolation using regression libraries with a tiny, fragmented dataset: projection of future values based on novel data patterns.

4.8 STM32Cube.AI

A software tool to port and optimize neural networks by processing libraries, code optimization, model conversion, and memory optimization. This tool can use any trained model by TensorFlow Lite, Keras, ONNX, Matlab, PyTorch, and Scikit learn [40]. STM32Cube.AI is able to convert the neural networks into an optimized code for appropriate MCU.

5. KEY ENABLERS OF TINYML

Several key enablers have contributed to the development and success of Tiny Machine Learning (TinyML). These technologies and methodologies play a crucial role in making machine learning models feasible and effective on resource-constrained devices.

There are currently a number of important gaps in TinyML research that need to be filled in order to improve its useful applications. Although TinyML performance has significantly increased due to developments in hardware optimization, such as energy-efficient microcontrollers and hardware accelerators, more hardware architectural optimization tailored to machine learning workloads on devices with limited resources is required. In a similar vein, methods like knowledge distillation, quantization, and pruning need to be improved in order to maintain model correctness while lowering computational requirements and size. Furthermore, transfer learning still has difficulties effectively adapting previously trained models for TinyML applications with confined data and processing power, despite its advantages in resource-constrained settings. Research is also required to create energy-efficient techniques for real-time jobs and optimize on-device inference for dynamic contexts. Additionally, the use of AutoML in TinyML offers a great chance to automate model design and optimization for edge devices with particular resource limitations.

Another important research gap in TinyML is privacy and security issues, particularly since these models frequently handle sensitive data. Secure and private techniques are required to protect data while inference and model training are taking place on edge devices. By addressing these shortcomings and creating secure, flexible, and lightweight methods, TinyML's full promise in practical applications can be realized. Future research should concentrate on developing hardware for specific TinyML activities, honing model size and performance optimization strategies, and strengthening security protocols to protect privacy in the expanding edge computing space. These developments will propel the creation of effective, perceptive, and privacy-aware TinyML solutions for a variety of sectors and uses.

5.1 Pruning

In machine learning and search algorithms, pruning is a data compression approach that decreases the size of decision trees by deleting parts of the tree that are unnecessary and redundant for classifying occurrences. Pruning lowers the final classifier's complexity, which increases predicted accuracy by reducing overfitting.

The weight in the LSTM layer is proposed to be compressed using a top-k weight row pruning with a quantization method, which will also make it easier to build effective hardware [41]. Because high weights matter more in LSTM network calculation than tiny weights, only the k greatest absolute values are maintained in each weight matrix row during pruning, and the remaining weights are set to zero. Even with iterative retraining, pruning causes an accuracy reduction of more than 4% when k is below 12, while significantly reducing the model size and, consequently, area and power consumption.

$\frac{k}{2}$ weight data is kept in the two weight partitions, input-to-hidden and hidden-to-hidden, using the standard top-k pruning. The improved technique, on the other hand, involves removing k1 and k2 weights from the input-to-hidden and hidden-to-hidden partitions, respectively, such that $k = k1 + k2$. When applying the enhanced top-k pruning, the accuracy is recovered from the standard top-k pruning.

The pruning approach can minimize model memory by deleting individual weights. However, by deleting too many edges, it is possible to lose what was learned. The

biggest challenge is determining the optimal model size. Iterative pruning is utilized because it outperforms one-time pruning [42]. However, pruning presents several technical challenges. Determining the optimal pruning ratio without degrading model performance is non-trivial, especially for sequential models like LSTMs that rely heavily on temporal information. Additionally, while iterative pruning—where pruning is applied gradually over the training epochs—has shown superior results compared to one-shot pruning, it requires more training time and introduces complexity in the optimization pipeline. To counteract the potential accuracy degradation, fine-tuning of pruned models is often necessary.

5.2 Quantization

Quantization is a process where we replace the floating-point numbers to an integer number. Quantization is the process of converting floating point numbers into discrete values so that they may be approximated by a series of integers and scaling factors [43]. As a result, the efficiency of the activities increases. This process is crucial to be used to reduce the memory requirement and power consumption of the model. A variety of ways are being used to find the best result for the quantization method toward the TinyML.

In a research paper [44], quantization-aware training is a method presented for the MobileNet hardware accelerator. Quantization-aware training is to quantize the parameters involved during training. Weights are quantized in this quantization due to the memory constraint. There are three formulas related to the quantized during training. The first step is a value pass a tanh function to produce an output range between -1 to 1 by using the following equation:

$$y\,(x_i) = \tanh(x_i) \tag{1}$$

Next, to remove negative values the values are biased and normalized as shown in equation 2.

$$q(y_i) = \frac{y_i}{2\max\,(y_i)} + \frac{1}{2} \tag{2}$$

Finally, the values are quantized based on the bit width, k using the uniform quantization method:

$$\hat{q}(q_i) = 2.\frac{1}{2^k - 1} round\; q\big((2^k - 1)q_i\big) - 1 \tag{3}$$

In the case of a convolutional neural network, the convolutional and fully connected layers require quantization-aware training for the weights. Please note that batch normalization layers also require quantization-aware training.

For batch normalization quantization, the O-means algorithm and linear transform are applied in the post-training quantization. The K-means algorithm divides all values into several groups using the cluster-center approach. The center value of the group is then used to replace the values that were previously grouped together. The linear transform method is discussed in this study. After determining the highest and lowest numbers, all values are then converted to the range from 0 to 1.

The mean, variance, beta, and gamma parameters of the batch normalization layer are quantized in this study using the K-means method and a linear transform. These parameters are further condensed to just two, and the process requires addition and multiplication, which can save half the amount of memory space and computational power.

Despite the advantages of quantization in reducing memory and power consumption, several challenges arise when applying it to TinyML models, particularly when working with deep learning architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). One of the primary challenges is maintaining model accuracy while reducing precision. As quantization transforms floating-point weights into lower-bit integer representations, the inherent loss of precision can lead to accuracy degradation, especially for complex models with intricate dependencies between parameters [44]. Quantization-aware training (QAT) attempts to mitigate this by incorporating the quantization process during training, but it adds significant complexity to the training pipeline and requires additional computational resources [45]. Model calibration is also a critical issue, as it involves determining the optimal scaling factors for weights and activations to minimize the quantization error. This is particularly challenging in environments where computational resources are constrained, and extra training steps can become prohibitively expensive [46].

Algorithmic optimizations are being actively developed to address these challenges. One promising technique is adaptive quantization, which adjusts the bit width dynamically based on the importance of each layer or parameter, allowing for more precision where necessary while still achieving significant reductions in size and computation[47]. Another approach, mixed-precision quantization, involves using lower-bit precision for less critical layers (such as activation layers) and higher precision for more critical weights (such as in the convolutional or fully connected layers).[48] Additionally, recent advancements in post-training quantization (PTQ) focus on reducing the need for training during the quantization process. PTQ techniques such as gradient-based optimization and clustered quantization leverage the optimization of quantization schemes to improve model accuracy without requiring the full retraining process [46]. Hardware-aware quantization, where quantization is tailored to specific edge-device architectures, is also becoming an essential area of research. These optimizations are crucial to enabling efficient and effective TinyML deployments on resource-constrained devices, balancing the trade-offs between model size, power consumption, and performance [49].

5.3 Knowledge Distillation

Knowledge distillation is a powerful technique and a key enabler for Tiny Machine Learning (TinyML). It is a process where a large, complex, and computationally expensive pre-trained model, often referred to as the

teacher model, is used to train a smaller, more lightweight model known as the student model. The student model aims to mimic the teacher model's behavior, learning from the teacher's knowledge and expertise.

During training, the student model picks up information from both the soft probabilities generated by the teacher model and the ground truth labels. The soft probabilities show how confident the instructor model is in its forecasts and offer further guidance for the student's learning process. By utilizing this transfer of information, the student model improves performance despite being smaller by gaining insights into the data's fine-grained patterns and linkages.

For TinyML, knowledge distillation offers a number of significant benefits. The student model is first compressed, greatly lowering its size and enabling deployment on edge devices with little memory and storage. Second, the faster inference speed produced by the compressed student model enables applications to respond instantly.

Furthermore, knowledge distillation makes sure that student model performance is equal to or even better than instructor model performance, preserving accuracy without compromising model quality. Additionally, because smaller models demand less processing power during inference, which is essential for battery-powered devices, the method helps minimize energy usage on edge devices.

Last but not least, knowledge distillation promotes privacy protection. The method ensures data privacy and security in TinyML applications by enabling the transfer of knowledge from the teacher model without disclosing sensitive or private data used in the teacher's training.

Overall, knowledge distillation strengthens TinyML through model compression, inference speedup, performance maintenance, energy reduction, and data privacy protection. It opens the door to the possibility of bringing advanced machine learning capabilities closer to users, boosting the development of intelligent and autonomous systems in a variety of real-world settings.

5.4 Hardware Advances

Hardware advances are instrumental in enabling the successful implementation of Tiny Machine Learning (TinyML) by addressing the unique challenges posed by resource-constrained devices. As the field of AI expands to the edge, where devices have limited computational power, memory, and energy resources, specialized hardware innovations have emerged to bridge the gap between machine learning and edge computing.

Energy-efficient microcontrollers have emerged as a cornerstone in the advancement of Tiny Machine Learning (TinyML). These small computing devices integrate a processor, memory, and peripherals into a single chip, making them ideal candidates for running machine learning algorithms on resource-constrained devices. In the past, microcontrollers struggled with limited computational capabilities, presenting a significant challenge for deploying complex machine-learning models. However, recent strides in microcontroller technology have led to the development of energy-efficient microcontrollers capable of handling machine-learning tasks while consuming minimal power. This crucial breakthrough has opened up new possibilities for bringing the power of AI to edge devices through TinyML.

Other than that, System-on-Chip (SoC) devices have been essential in encouraging TinyML's expansion. SoC devices offer an integrated solution that is well suited for IoT applications by merging several components, including processors, memory, communication interfaces, and sensors, into a single chip. SoC devices are ideally suited for TinyML deployments as a result of considerable improvements in power efficiency and processing power over time. They have successfully integrated machine learning models into edge devices thanks to their smooth integration of several functionalities onto one small chip, enhancing the IoT landscape with intelligent, real-time applications.

Moreover, the performance of edge devices has been further improved by hardware accelerators, which have emerged as a game-changer in the field of Tiny Machine Learning. Key machine learning tasks like matrix multiplications and convolutions may be accelerated by specialized hardware accelerators like Tensor Processing Units (TPUs) and Neural Processing Units (NPUs). These accelerators enable real-time processing of complicated data streams by drastically speeding up machine learning inference on edge devices. TinyML applications may attain astounding levels of effectiveness and responsiveness with the use of hardware accelerators, enabling edge devices to carry out intelligent activities without human intervention.

TinyML implementations have greatly benefited from improvements made to their overall effectiveness thanks to low-power co-processors. These auxiliary processors are made to delegate particular duties to the primary processor, thus lightening the computing load and conserving energy. Low-power co-processors have been designed specifically for machine learning activities to maximize performance and reduce power consumption, especially for energy-constrained devices. Without sacrificing power efficiency, their incorporation into edge devices has considerably aided in the effective deployment of machine learning models across a range of industries, from smart homes to industrial automation.

As a result, in order to accommodate TinyML models inside the memory restrictions of resource-constrained devices, memory optimization has emerged as a critical component. Running complex algorithms for machine learning on edge devices is difficult due to their small memory capacities. The memory footprint of TinyML models has been greatly decreased, nevertheless, because of improvements in-memory technology and optimization methods. These improvements make it possible for machine learning algorithms to function well even on hardware with low memory, further demonstrating TinyML's suitability for use in a variety of edge computing applications.

The application of Tiny Machine Learning on edge devices requires effective power management strategies. To optimise energy usage on edge devices, strategies like dynamic voltage and frequency scaling (DVFS) and sleep modes have been developed. By balancing performance

and energy economy, these power management techniques enable devices to dynamically adapt their performance and power use based on workload. TinyML apps may extend battery life and provide smooth, real-time operations on devices with limited resources by effectively managing power resources.

Tiny Machine Learning on edge devices has been strengthened by the addition of specialized Edge Processing Units (EPUs). These EPUs were created specifically to execute machine learning workloads quickly. EPUs improve the overall performance of edge devices by meeting the computing needs of TinyML tasks, allowing them to execute complicated AI tasks with extraordinary efficiency. TinyML's deployment on edge devices has thus become smoother, encouraging the development of intelligent, autonomous applications that make the most of edge computing's capabilities.

The potential for implementing machine learning on edge devices has been completely transformed by the constant growth of hardware improvements. Tiny Machine Learning may now be applied to a variety of products, including wearables, smart home appliances, industrial sensors, and autonomous systems. Tiny Machine Learning will perform better and be more applicable in a variety of areas as hardware advances in terms of energy efficiency, processing power, and memory capacity.

5.5 On-device Inference

A key component of Tiny Machine Learning (TinyML), on-device inference, is essential for implementing effective and responsive machine learning applications on devices with limited resources. On-device inference provides a number of benefits by allowing machine learning models to be performed directly on edge devices without requiring ongoing contact with cloud servers, making it a crucial enabler for TinyML applications.

Reduced latency is one of the main advantages of on-device inference. Faster response times occur from the elimination of the requirement to transport data to remote servers for processing when data processing is done locally. For time-sensitive applications, such as autonomous cars, gesture recognition systems, or voice assistants, where quick choices and actions are essential, this real-time or near-real-time capacity is very beneficial.

On-device inference also improves security and privacy. Sensitive information does not leave the user's control during inference since it stays on the device and is not sent to other servers. This is crucial for applications that deal with sensitive information or personal data, such as smart home devices or healthcare monitoring systems, where data privacy is of the highest importance.

Applications may run without an internet connection thanks to on-device inference, which also supports offline operation. In situations when connectivity is sporadic or nonexistent, this functionality is useful since it ensures uninterrupted functioning.

Improved energy efficiency is a key benefit of on-device inference. On-device inference minimizes the energy required for transmission by eliminating the need to send data to the cloud for processing. Edge devices benefit from extended battery life as a result, which increases their sustainability and efficiency over time.

Examples of on-device inference in TinyML applications abound across various domains. In speech recognition, the model can be deployed on a smartphone or smart speaker to interpret and understand spoken commands locally, providing real-time responses to users without relying on cloud-based processing. In cameras and surveillance systems, on-device inference can be employed for real-time object detection, identifying and categorizing objects in the camera's field of view without sending the entire video stream to a cloud server for processing. Wearable devices can use on-device inference to recognize and classify users' activities, such as walking, running, or sleeping, directly on the device without requiring a constant connection to the cloud.

On-device inference allows for predictive maintenance in industrial environments, as TinyML models installed on edge devices examine sensor data in real-time to identify probable faults or maintenance requirements, improving the effectiveness and dependability of industrial systems. On-device inference is, in general, a key component of tiny machine learning, enabling edge devices to make deft decisions locally, decreasing reliance on cloud resources, and opening up a wide range of applications that need real-time responsiveness, enhanced privacy, and energy efficiency.

Furthermore, by reducing the possible effects of network downtime or latency problems, on-device inference improves the dependability and resilience of TinyML systems. When local processing is used, edge devices may carry out crucial functions without interruption even when internet access is spotty or nonexistent. Because of its robustness, on-device inference is especially useful in disconnected or challenging settings, such as in applications for agriculture, environmental monitoring, or disaster response, where dependable connectivity cannot be assured.

Additionally, the direct deployment of machine learning models on edge devices enables more effective network bandwidth usage. On-device inference lowers data transmission costs and aids in overall network optimization by eliminating the need to send massive volumes of data to the cloud for processing. This makes the adoption of TinyML solutions more affordable, which can be particularly helpful in situations where network bandwidth is constrained or expensive.

On-device inference solves privacy issues by keeping sensitive data localized to the device in addition to its technological benefits. This independent strategy provides more data privacy and protection, in line with user expectations and data privacy laws. Therefore, on-device inference encourages customer acceptance and confidence in TinyML applications, enabling a wider deployment of these technologies in consumer-centric domains such as personal assistants, wearable health monitoring devices, and smart home appliances.

5.6 Transfer Learning

Transfer learning is a powerful and essential technique that has significantly contributed to the success of Tiny

Machine Learning (TinyML). Its ability to leverage knowledge gained from pre-trained machine learning models on a source task and apply it to a related target task with limited data makes it particularly valuable in resource-constrained environments. In the context of TinyML, where collecting vast amounts of labeled data for training is often impractical, transfer learning offers an efficient solution to build accurate and robust models.

The process of transfer learning begins with training a machine learning model on a large dataset for a source task. For example, a convolutional neural network (CNN) might be trained on a diverse collection of general images for image classification. During this phase, the model learns generalized features, patterns, and representations from the extensive dataset. This knowledge serves as a foundation for the subsequent target task.

The model is adjusted or adapted to the target task with little data once it has been pre-trained on the source task. In order to match the model's parameters and weights with the particulars of the target task's specifications, fine-tuning involves performing such adjustments. A small portion of the data required for training from scratch is used to start the learning process for the target task using the generalized knowledge of the pre-trained model.

Data efficiency is one of transfer learning's most important benefits when used with TinyML. Large volumes of labeled data are often needed to train machine learning models from the start in order to reach respectable accuracy. It might not be possible to gather such huge datasets on devices with limited resources. By utilizing pre-trained models, transfer learning gets over this restriction and enables TinyML applications to operate well with a lot less target task data.

The influence on training time of transfer learning in TinyML is another significant advantage. It may be time-consuming and computationally costly to train a machine-learning model from the start. TinyML models can be deployed on edge devices more quickly since the fine-tuning procedure only needs a few epochs when working with a pre-trained model.

Additionally, the model performs better in the target job thanks to the generalized information it gained during the source task. The target task's domain is better understood by the model, which enhances its accuracy and adaptability. This feature is especially useful in edge computing applications where precision and real-time responsiveness are essential components.

Different domains are represented by transfer learning examples in TinyML. A pre-trained CNN model in image recognition may be adjusted to carry out certain tasks on edge devices, including categorizing photos related to a particular context or recognizing objects in a given domain. For instance, a CNN model may be fine-tuned to recognize particular items pertinent to farm or industrial settings on edge devices after being trained on an extensive database of general photos.

In natural language processing, pre-trained language models, such as BERT or GPT, can serve as starting points for various tasks, with fine-tuning on domain-specific data. For example, a pre-trained language model can be fine-tuned to perform sentiment analysis on customer reviews or intent detection in customer service interactions, enhancing the capabilities of TinyML-based chatbots or virtual assistants.

Similarly, transfer learning can be applied to audio-related tasks in TinyML, such as speech recognition or environmental sound classification. An audio model that has already been trained may be fine-tuned to recognize certain spoken commands on edge devices, enabling voice control of smart home appliances or hands-free mobile app usage.

In conclusion, transfer learning is essential for TinyML to succeed in situations with limited resources. TinyML applications can achieve accuracy and efficiency even with small quantities of data by using information from pre-trained models. As a result, edge devices are given the ability to be intelligent and adaptable, making transfer learning a key facilitator for delivering AI to people in a variety of real-world circumstances.

5.7 AutoML

Automated machine learning, or AutoML, is a ground-breaking innovation that simplifies and automates key machine learning processes. It aspires to make AI accessible to a wider audience, including people who lack in-depth knowledge of data science or machine learning, by streamlining the difficult and time-consuming activities involved in creating, training, and deploying machine learning models.

Feature engineering, model selection, hyperparameter tweaking, and assessment are a few manual stages that are frequently included in the development of a machine-learning model. For many people and organizations, this work requires a high level of topic knowledge and skill, which can be a barrier to acceptance. By automating these procedures, AutoML aims to reduce these difficulties, allowing users to concentrate on specifying the issue and supplying the data while the system handles the rest.

The potential of AutoML to accelerate the model creation process is one of its main benefits. A machine learning model must typically be created from scratch, which can take a lot of time and need several cycles of testing and fine-tuning. The model may be repeatedly improved until it performs at its best using the most appropriate algorithms and hyperparameters selected by autoML algorithms, which can automatically search and explore the model architecture space. The time and effort needed to create high-performing models are considerably reduced by this automated search procedure.

Additionally, by making machine learning usable by non-experts, AutoML contributes to its democratization. Without an extensive understanding of data science or programming, data scientists, developers, and domain experts with little machine learning ability can use AutoML platforms to generate complex models. Through the democratization of machine learning, a wider variety of people and organizations are now able to use AI and machine learning capabilities in their workflows and applications.

Additionally useful for managing complex model settings and hyperparameter modifying is AutoML. For machine learning to function at its best, choosing the

appropriate model architecture and fine-tuning hyperparameters are essential. These configurations can be labor- and time-intensively searched for manually. In order to find the optimal combinations for the given dataset and job, AutoML's automated search algorithms can effectively examine a large variety of options. This eventually improves model performance.

AutoML also deals with the problem of model repeatability. It might be difficult to replicate an exact model and results when using traditional machine learning techniques because they are frequently iterative. The entire process is automated and documented with AutoML, making it possible to simply repeat the same model, improve teamwork, and enable reproducible research.

Despite the fact that AutoML has many advantages, it is important to understand its limits. AutoML is not a one-size-fits-all approach, thus it might not always be appropriate for highly specialized or domain-specific issues that call for specialized expertise. AutoML's totally automated nature may also lead to reduced interpretability and transparency in model decision-making, which can be problematic in sectors or applications that need strict compliance with rules.

6. USE CASES, DATASET & MODEL

6.1 Use Case

TinyML has a broad range of potential applications that demonstrate its ability to perform advanced machine learning tasks on small, low-power devices. For instance, it can be used in speech recognition systems to enable voice-based commands on devices like smartphones and smart speakers. Keyword spotting is another application, where TinyML can detect specific words or phrases in audio streams, enabling hands-free interactions with devices. Audio wake word detection is similar, allowing devices to stay in a low-power mode until they hear a trigger word, which is useful for voice-activated systems.

In the realm of visual recognition, image recognition enables devices to identify objects, faces, or scenes, which is important for applications like security cameras or smart home systems. Hand gesture recognition and sign language translation are also made possible by TinyML, allowing for intuitive interaction with devices and improving accessibility for individuals with disabilities. Anomaly detection is another area where TinyML can be applied, helping to monitor systems and detect unusual behavior or potential security threats in real-time.

Finally, TinyML can support autonomous systems by processing sensor data for tasks like object detection in self-driving vehicles, or for monitoring the environment in various applications like agriculture or wildlife conservation. These use cases highlight the versatility of TinyML in bringing advanced machine learning capabilities to edge devices with limited resources, enabling smarter, more efficient, and responsive systems.

6.1.1 Speech Recognition

Speech recognition is the general application of machine learning. One of the research projects for 2020, speech recognition is developed for low-end devices called EdgeRNN, a small voice recognition network containing spatiotemporal characteristics for use in edge computing [50]. EdgeRNN uses a 1-Dimensional Convolutional Neural Network (1-D CNN) and Each frequency domain of the auditory characteristics' temporal information is processed by a recurrent neural network (RNN). In order to increase the percentage of the network that contributes to the identification, a simplified attention method was applied. The effectiveness of EdgeRNN has been evaluated overall in terms of speech emotion and keyword recognition. Speech emotion recognition uses the IEMOCAP dataset, and the unweighted average recall (UAR) is 63.98%. With a weighted average recall (WAR) of 96.82%, Google's Spoken Commands Datasets V1 are used for speech keyword recognition. The accuracy of EdgeRNN has increased when compared to the trial outcomes of the related efficient networks on the Raspberry Pi 3BC for voice emotion and keyword recognition.

6.1.2 Keyword Spotting

KWS may be characterized as a task that extracts keywords from voice sounds. Keyword spotting system with pruning and quantization method develop a compact KWS system, which uses an STM32F7 microcontroller with a Cortex-M7 core running at 216 MHz and 512 KB of static RAM [51]. In order to fulfill the constraint of edge devices, the convolutional neural network (CNN) architecture has reduced the number of operations for KWS. In KWS, it consists of feature extraction, neural networks, and posterior handling. Every 37 milliseconds, our baseline system produces classification results, including real-time audio feature extraction. Four keywords are chosen from the Speech Command Data Set v0.01 [14]: "yes," "no," "left," and "right."

6.1.3 Audio Wake Word

Audio wakeword is a system that can detect a phrase in an audio. Similar to speech recognition and keyword spotting, the machine learning model for this system requires large memory to operate at first. After a few years, the evolution of the system takes place to improve the existing models. Tiny Convolutional Recurrent Neural Network (Tiny-CRNN) was proposed to improve the problem in wakeword detection and augmented using scaled dot-product attention. The aim of building this model is to minimize the number of False Accepts.

6.1.4 Image Recognition

AttendNets [52] was introduced as a low-precision and compact neural network for on-device image recognition. AttendNets possess deep self-attention architectures based on visual attention condensers, which extend on the recently introduced standalone attention condensers to improve spatial-channel selective attention. Furthermore, AttendNets have unique machine-designed macro architecture and microarchitecture designs achieved via a machine-driven design exploration strategy. AttendNet outperforms MobileNet-V1 by 7.2% while using 3.0 fewer

multiply-add operations, 4.17 fewer parameters, and 16.7 percent less weight memory. Based on these encouraging outcomes, AttendNets shows how visual attention condensers work as the foundation for a variety of on-device visual perception tasks for TinyML applications.

6.1.5 Hand Gesture Recognition

Hand gesture recognition is a system that recognizes a hand gesture from an image without using any image region selection framework. For this system, CNN was proposed to detect the hand gesture in one of the research projects based on the hand region shrinking and overfitting [53]. To get a high-performance recognition system in a scenario where the pattern to be detected (the hand) is relatively small in comparison with the image size, it is very important to extract high-level abstract features from raw data. It is well-known that deeper networks allow us to learn more powerful semantic representations. Therefore, the number of convolutional layers and pooling layers should be large enough for the ongoing recognition task. However, the reduced size of the hand inside the image produces a shrinking problem according to the data traveling deeper and deeper across the network. The effective activated area containing relevant hand information for classification tasks shrinks exponentially with the increasing number of layers. Overfitting is a common problem in machine learning. For the proposed hand recognition task, the region of interest is relatively small, causing misleading behaviors in CNN learning, such as trying to infer the hand gesture from non-related image areas (for example the human body or the human face). Therefore, overfitting becomes a major problem in such conditions. To prevent this kind of behavior, the CNN design should restrict the number of network parameters. For this purpose, small convolutional filter sizes of 3x3 pixels have been adopted, except for the first layer which is 5x5 pixels. This leads to a significant improvement in accuracy in comparison with larger filter sizes. Additionally, a dropout learning strategy has been applied to the fully connected layers with the same purpose of preventing overfitting. The designed hand-gesture recognition network can classify seven sorts of hand gestures in a user-independent manner and in real-time, achieving an accuracy of 97.1% in the dataset with simple backgrounds and 85.3% in the dataset with complex backgrounds.

6.1.6 Sign Language

American Sign language (ASL) is amongst one of the most widely used sign languages in the world and some estimates indicate it is used by anything between 250 and 500 thousand signers in the USA itself. ASL is broadly composed of three major classes: fingerspelling (each alphabet has a symbol), word level vocabulary (each word has a symbol), and non-manual features (the signer uses an amalgamation of facial expressions, mouth, tongue, and body poses to communicate).

In this research [54], an extremely efficient, real-time, well-generalizable CNN-based solution for ASL fingerspelling recognition has been implemented on an ARM Cortex-M7 powered OpenMV H7 board, which is just 185 KB in size and the inference speed is 20fps. The method proposed in this paper is highly generalizable and is applicable to any ARM microcontroller and can also be easily ported to other architectures.

Four datasets were used in this research work. The first dataset was the Sign MNIST dataset from Kaggle [55]. It was created from a set of 1704 uncropped color images taken in various backgrounds. The second dataset was produced by modifying the Kaggle ASL Kaggle [55]. having 87000 images of dimension 200x200 for 29 classes, the 26 English alphabets, and 3 classes for Space, Nothing and Delete. This paper produced the third dataset using the target OpenMV H7 camera [54]. The images taken were of 240x240 size and had approximately 400 images for each of the 24 classes, they were downscaled to 28x28 using the inter_area interpolation in OpenCV. The images were shuffled and split into 10% for training and 90% for testing. The fourth dataset was the Kaggle ASL Alphabet Test dataset [56], created by Dan Rasband, in order to test the effectiveness of the models in real-world scenarios with noisy backgrounds.

It has 98.84% test accuracy and 74.58% generalization accuracy. It has been shown that interpolation, which is a normal step in image resizing, might be used to reduce accuracy drop during full integer quantization, simultaneously improving model generalizability, and that model performance is best if inputs and outputs are in float32.

6.1.7 Anomaly Detection

An Intrusion Detection System (IDS) is a mechanism that detects intrusions or attacks against a system or a network by analyzing the activity of the network and the system. Such intruders can be internal or external where the internal intruders are users inside the network that attempt to raise their access privileges to misuse non-authorized privileges while external intruders are users outside the target network attempting to gain unauthorized access to the network. The IDS monitors the operations of a host or a network, alerting the system administrator when it detects a security violation.

In this paper [57], we have proposed a lightweight intrusion detection model based on machine learning techniques. This model can detect new attacks and provide double protection to the IoT nodes against internal and external attacks. In order to find the best classifier model, we evaluated several machine learning classifier models using three lightweight feature selection algorithms and tried to optimize the parameters of each algorithm to get an efficient classifier model with high accuracy and precision, as well as low false negatives. In the experiments, we used KDD99, NSL-KDD, and UNSW-NB15 datasets to learn and evaluate our model.

According to the results of our study, it is observed that DT and KNN performed better than the other algorithms; however, the KNN takes much time to classify compared to the DT algorithm. Furthermore, with the three correlation methods used to reduce dataset dimensions such as PCC, SCC, and KTC, the classifiers produce good performance when the threshold of the correlation coefficient is greater than 0.9; below this threshold, performances are poor and sometimes unacceptable. In the case of the datasets that relate to the extent of our study area, it is found that the performance obtained on the NSL-KDD dataset is better compared to the KDD99 and UNSW-NB15 datasets.

6.1.8 Autonomous Vehicle

In the era of technology and revolution, the invention of autonomous vehicles has started step by step. For autonomous driving, the crucial part is the ability to detect an object such as a vehicle, pedestrian, divider, and others. Therefore, the detection system must have high accuracy. Furthermore, memory consumption for each task and speed are the challenges that should be considered in creating an autonomous driving system. In this paper [58], we particularly implement and evaluate meta-architectures based on Faster R-CNN, R-FCN, and SSD along with selective feature extractors such as ResNet50, ResNet101, MobileNet_V1, MobileNet_V2, Inception_V2, and Inception_ ResNet_V2. These models are trained on the MS COCO database. To better reveal their performance in vehicle and pedestrian detection, we fine-tune such models on the KITTI data set using a transfer learning method.

We report the extensive evaluation results of these detectors in terms of accuracy, execution time, memory consumption, number of FLOPS, and parameter size. We conclude that Faster R-CNN ResNet 50 achieves the highest AP and the best balance between processing speed and accuracy. Faster R-CNN Inception_V2 performs best in detecting cars or detecting pedestrians. SSD MobileNet V2 is the fastest model, and SSD MobileNet V1 is the lightest model, both of which are suitable for applications on mobile and embedded devices.

6.2 Dataset

There are a number of open-source datasets that are relevant to TinyML use cases. Table 2 breaks them down by the type of data. Despite the availability of these datasets, the majority of deployed TinyML models are trained on much larger, proprietary datasets. The open-source datasets that are competitively large are not TinyML-specific. The lack of large, TinyML-focused, open-source datasets slows the progress of academic research and limits the ability of a benchmark to represent real workloads accurately.

In the the context of TinyML, datasets need to be small, efficient, and task-specific to accommodate the limited memory, processing power, and storage of edge devices like microcontrollers. Unlike conventional machine learning, which can rely on large, general-purpose datasets and high-resolution data, TinyML applications typically require lower-resolution, compressed data formats and minimal preprocessing to ensure real-time performance on resource-constrained hardware. Additionally, datasets for TinyML often need to be privacy-aware and optimized for on-device use, which limits the direct applicability of many open-source datasets designed for traditional cloud-based ML.

Table 1. TinyML use cases, models, and datasets

Type	Use case	Model	Dataset
Audio	Speech recognition, Keyword spotting, Audio wakeword	CNN RNN DS-CNN	Google's speech command (65000 words), Visual wake word
Image	Image recognition, Hand gesture recognition, Sign language	CNN Visual attention condenser	Kaggle Sign MNIST (28000 images), Kaggle ASL (87000 images), Kaggle ASL Alphabet (87000 images)
Industry	Anomaly detection, Autonomous vehicle	KNN DT Faster R-CNN R-FCN SSD	KDD Cup 99 (4.9M data) NSL-KDD (125973 data) UNSW-NB15 (2.5M data) Microsoft-COCO (330000 images) KITTI (7000 images)

6.3 Model

Table 2 lists common model types for TinyML use cases. A significant change has occurred with the recent achievability of machine learning on Microcontroller Unit (MCU)-class devices. In contrast to the broad adoption of MobileNets in the mobile device space, the TinyML environment presently lacks widely accepted models. Due to the lack of standardized benchmarks, this absence makes it difficult to choose appropriate models for deployment on MCU-class devices.

This relative immaturity, nevertheless, also represents a chance. Our choices and contributions within this developing sector can have a significant impact on TinyML's future development and direction. It enables us to set benchmarks, direct optimization techniques, and have an impact on the creation of model designs.

Selecting a subset of the currently available models, outlining the rules for quality versus accuracy trade-offs, and prescribing a measurement methodology that can be faithfully reproduced will encourage the community to

develop new models, runtimes, and hardware that progressively outperform one another. Enhancing model performance while maintaining a balance between memory utilization, speed, and accuracy on devices with limited resources is the research gap in TinyML applications, such as autonomous vehicles, speech recognition, and picture recognition. Despite improvements such as EdgeRNN and Faster R-CNN, problems with model size, inference speed, and efficiency still exist. More effective training techniques and model designs are also required, as evidenced by problems like overfitting in gesture recognition and the requirement for improved generalization in sign language recognition. Additionally, to maximize speed without sacrificing real-time capabilities, methods like quantization, pruning, and transfer learning must be effectively integrated.

7. CHALLENGES

TinyML is a new and rapidly developing field of research that has the potential to revolutionize the way we interact with intelligent devices. However, this technology faces several challenges, including the development of efficient machine-learning algorithms, the design of low-power computer architectures, and the need to ensure security and privacy in the deployment of machine-learning models on edge devices. Despite these challenges, the potential benefits of TinyML are significant, and many researchers and industry experts are working hard to overcome these obstacles and unlock the full potential of this exciting new field.

7.1 Power Consumption

The primary focus of research in Tiny Machine Learning (TinyML) revolves around addressing the limited power resources available in edge devices. TinyML models require a certain power threshold to maintain the accuracy of their algorithms. However, specifying the power consumption is challenging due to the diverse designs and preprocessing methods employed in various hardware platforms. Additionally, as edge devices often incorporate sensors and additional connections, TinyML systems may encounter power consumption issues. Consequently, the design of an energy-efficient TinyML system remains a significant challenge. Energy efficiency is critical for edge devices, as they often run on limited power sources. TinyML models must be designed to minimize power consumption while performing complex computations.

7.2 Small Memory

Edge devices, such as microcontrollers, IoT sensors, and wearable devices, often have limited memory or RAM compared to traditional computing systems. This limited memory can range from a few kilobytes to a few megabytes, significantly smaller than the memory available in conventional computing systems. The size of TinyML models may exceed the available memory on edge devices. Machine learning models, even when optimized for edge deployment, can still have substantial memory requirements. Neural network architectures or even simpler machine learning algorithms can demand significant memory for storing model parameters. Small memory on the device presents a challenge in deploying the neural networks model. The solution to this problem is to combine hardware improvements, memory management techniques, and model optimization approaches to make sure that TinyML models can function well within the constrained memory resources on edge devices.

7.3 Limited Datasets

There are a lot of popular public datasets available for training machine learning models but not many available to train ultralow-power models for embedded systems. TinyML models heavily rely on high-quality and diverse datasets for training. TinyML architecture may not be suitable for an existing dataset because of its low power consumption, which limits its flexibility. In an ideal world, these datasets would have the temporal and spatial resolution, noise level, and diversity of data that are expected from sensors connected to low-power embedded devices. Limited datasets can lead to inadequate model performance, reduced accuracy, or biased predictions.
. We underline the requirement for a common set of datasets that can be easily taken into account for training the TinyML systems in this way is important for the systematic development of best practices and methods for embedded inference and system performance benchmarking.

7.4 Adaptability and Flexibility

Most of the time, software updates are rare for low-power MCU-powered devices. These devices continue to use the same software after being programmed. However, the deployed inference models should adapt and take into account new conditions when these devices collect data of new input that was not included in the first training dataset. The requirement for constant model updating leads to an issue on how to consider new knowledge into the model and how to actually update the model, as updating the software on several devices can be difficult and sometimes impossible.

7.5 Data Collection and Labeling

Collecting sufficient amounts of diverse and high-quality data, especially for specialized applications or rare scenarios, can be challenging. Limited data availability can hinder the training process and model performance. Acquiring huge datasets can be difficult in edge computing contexts where TinyML runs because of restricted connection, privacy issues, or energy limits that limit data gathering. It's critical to ensure accurate and correct labeling of datasets. Manual labeling, while prone to errors and biases, can be costly, time-consuming, and may introduce inaccuracies or biases into the training set. TinyML models can be affected by biassed or insufficiently labeled data, which can lead to models with poor generalization or performance in real-world applications.

7.6 Security and Privacy

User privacy is an issue when processing data locally on edge devices since personal data may be visible or

susceptible to privacy violations if improperly protected. Insecure data handling practices can compromise the integrity of the training data or models, leading to biased or unreliable predictions and reduced model effectiveness. Balancing security requirements with the need to keep the TinyML models and processes lightweight and efficient is crucial. Adding security measures can increase complexity and resource usage. It can be difficult to implement strong security measures on edge devices with limited resources without appreciably affecting performance or energy consumption.

8. FUTURE INSIGHT

The potential of Tiny Machine Learning (TinyML) is truly remarkable and is expected to revolutionize several industries in the near future. As TinyML is gaining popularity, more and more businesses are recognizing its benefits and incorporating AI on edge devices with constrained resources. In situations where real-time responsiveness, privacy, and low latency are crucial factors, the ability to process data locally without over-reliance on cloud resources will become increasingly important. This development can lead to exciting new possibilities and innovations in the field of AI and edge computing.

8.1 Hardware and Application

TinyML applications will be more precise and effective thanks to these models, which are created to optimize performance and energy efficiency for certain activities in industries like healthcare, agriculture, industrial automation, and others.

Hardware improvements will continue to be crucial in supporting TinyML. It is anticipated that future developments in energy-efficient microcontrollers, specialized accelerators, and edge processing units will provide edge devices that are even more powerful and power-efficient.

8.2 Model Development

TinyML-specific automatic machine learning (AutoML) methods will progress. The development and deployment of machine learning models on edge devices will become simpler for non-experts as a result. Developers and subject-matter experts will be able to concentrate on problem-solving rather than model optimization thanks to AutoML, which will streamline the model creation process.

In TinyML, federated learning, which trains models locally on edge devices and updates them cooperatively without transferring raw data, may find success. With this method, concerns about data privacy are addressed, and sensitive data is kept local while edge devices collaborate to improve their models.

The deployment of TinyML in remote or difficult-to-reach places will be made possible through research and advancements in energy harvesting technologies in terms of sustainability and energy gathering. TinyML applications will have a wider audience thanks to energy-efficient edge devices and sustainable energy sources, especially for environmental monitoring and conservation projects.

8.3 System Architecture

The edge-to-edge collaboration will spread more widely as interconnected edge devices communicate and work together to solve shared problems. This kind of distributed coordination will enable more advanced and seamless applications across devices, significantly enhancing the capabilities of TinyML systems.

TinyML models may become adaptable over time and constantly enhance their performance. In order to continuously improve AI capabilities without the need for human updates, models might update and fine-tune themselves on the edge using real-time feedback from users and data sources.

8.4 Research and Collaboration

Finally, there will be more multidisciplinary study in the field of little machine learning. Innovative solutions that solve specific issues in TinyML development and deployment will result from collaboration between AI researchers, hardware engineers, data scientists, and domain specialists.

8.5 Impact and Vision

Tiny Machine Learning has a bright future, thanks to a growing variety of applications and technological breakthroughs that make AI more usable and potent on edge devices with limited resources. TinyML's continuing development will open the door for intelligent, autonomous systems that will bring the advantages of artificial intelligence to everyday life, enabling a variety of businesses and having a good effect on society.

9. CONCLUSION

The review paper on Tiny Machine Learning has provided valuable insights into the emerging field of machine learning on resource-constrained devices. Despite being still in its early stages, Tiny Machine Learning (TinyML) is rapidly advancing and offers access to innovative industrial applications. To ensure universal acceptance of TinyML implementations, it is essential to manage the range of hardware and software through benchmarking standards. The development of compact machine learning, part of the edge-IoT toolchain, aims to seamlessly integrate TinyML into various edge devices. A paradigm change in TinyML capabilities is possible by emphasizing in-memory processing and neuromorphic computing. Continuous updating of benchmarks with fresh datasets is essential for smooth interactions with low-power embedded devices. Community involvement from organizations and enthusiasts plays a vital role in improving datasets, creating new benchmarks, and refining TinyML procedures. Therefore, the rapid progress of TinyML, along with efforts to address hardware and software diversity, signifies a transformative future with the potential to reshape workplace technology and increase industrial efficiency. Embracing TinyML will open up new and exciting possibilities for smart handheld devices, microcontrollers, IoT-edge systems, and more, giving rise

to a more intelligent and connected world. The path to a fully developed TinyML ecosystem offers a lot of potential for the development of edge computing and AI in the future.

ACKNOWLEDGMENT

This work has been supported by Universiti Teknologi Malaysia under Graduate Research Assistant (GRA) (R.J130000.7623.4C815).

REFERENCES

[1] L. Ravaglia, M. Rusci, D. Nadalini, A. Capotondi, F. Conti, and L. Benini, "A TinyML Platform for On-Device Continual Learning with Quantized Latent Replays," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 11, no. 4, pp. 789–802, 2021, doi: 10.1109/JETCAS.2021.3121554.

[2] E. Lattanzi, M. Donati, and V. Freschi, "Exploring Artificial Neural Networks Efficiency in Tiny Wearable Devices for Human Activity Recognition," *Sensors*, vol. 22, no. 7, 2022, doi: 10.3390/s22072637.

[3] G. Muhammad and M. S. Hossain, "Emotion Recognition for Cognitive Edge Computing Using Deep Learning," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 16894–16901, 2021, doi: 10.1109/JIOT.2021.3058587.

[4] M. Goudarzi, M. S. Palaniswami, and R. Buyya, "A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments," *IEEE Trans. Mob. Comput.*, vol. 1233, no. XX, 2021, doi: 10.1109/TMC.2021.3123165.

[5] J. Kwon and D. Park, "Hardware/software co-design for tinyml voice-recognition application on resource frugal edge devices," *Appl. Sci.*, vol. 11, no. 22, 2021, doi: 10.3390/app112211073.

[6] S. Disabato, M. Roveri, and C. Alippi, "Distributed Deep Convolutional Neural Networks for the Internet-of-Things," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1239–1252, 2021, doi: 10.1109/TC.2021.3062227.

[7] A. Parodi, F. Bellotti, R. Berta, and A. De Gloria, "Developing a machine learning library for microcontrollers," *Lect. Notes Electr. Eng.*, vol. 550, no. 9783030119720, pp. 313–317, 2019, doi: 10.1007/978-3-030-11973-7_36.

[8] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4403–4417, 2020, doi: 10.1109/JIOT.2020.2976702.

[9] C. Banbury *et al.*, "MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers," no. Figure 1, 2020, [Online]. Available: http://arxiv.org/abs/2010.11267.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1026–1034, 2015, doi: 10.1109/ICCV.2015.123.

[11] A. Osman, U. Abid, L. Gemma, M. Perotto, and D. Brunelli, "TinyML Platforms Benchmarking," *Lect. Notes Electr. Eng.*, vol. 866 LNEE, pp. 139–148, 2022, doi: 10.1007/978-3-030-95498-7_20.

[12] G. Cornetta and A. Touhafi, "Design and evaluation of a new machine learning framework for iot and embedded devices," *Electron.*, vol. 10, no. 5, pp. 1–42, 2021, doi: 10.3390/electronics10050600.

[13] M. T. Lê, P. Wolinski, and J. Arbel, "Efficient Neural Networks for Tiny Machine Learning: A Comprehensive Review," pp. 1–39, 2023, [Online]. Available: http://arxiv.org/abs/2311.11883.

[14] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017, [Online]. Available: http://arxiv.org/abs/1704.04861.

[15] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 6848–6856, 2018, doi: 10.1109/CVPR.2018.00716.

[16] S. Soro, "TinyML for Ubiquitous Edge AI," no. 20, 2021, [Online]. Available: http://arxiv.org/abs/2102.01255.

[17] C. R. Banbury *et al.*, "Benchmarking TinyML Systems Challenges and Direction," 2020.

[18] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello Edge: Keyword Spotting on Microcontrollers," pp. 1–14, 2017, [Online]. Available: http://arxiv.org/abs/1711.07128.

[19] A. Edgebadge and T. Lite, "Adafruit EdgeBadge - TensorFlow Lite for Microcontrollers." https://www.adafruit.com/product/4400.

[20] Ambiq Micro, "Apollo3 Blue - Ultra-low Power MCU," 2019. https://cdn.sparkfun.com/assets/learn_tutorials/9/0/9/Apollo3_Blue_MCU_Data_Sheet_v0_9_1.pdf.

[21] S. F. Barrett, "Arduino Nano 33 BLE Sense," *Synthesis Lectures on Digital Circuits and Systems*, 2023. https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf.

[22] "Arduino® Portenta H7 Collective Datasheet," 2021. https://docs.arduino.cc/resources/datasheets/ABX00042-ABX00045-ABX00046-datasheet.pdf.

[23] N. X. P. Semiconductors, "FRDM-K64F: Freedom Development Platform for Kinetis® K64, K63, and K24 MCUs," 2018. https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/k-seriesperformancem4/k2x-usb/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F.

[24] C. Lake, "Gap8," 2021. https://en.wikichip.org/wiki/greenwaves/gap8.

[25] "GAP9." .

[26] P. R. Manual, "Arduino ® Nicla Sense ME Target areas : Arduino ® Nicla Sense ME Features," 2023. https://docs.arduino.cc/resources/datasheets/ABX00050-datasheet.pdf.

[27] NORDIC Semiconductor, "nRF52840 Development Kit Schematic," 2019. https://infocenter.nordicsemi.com/pdf/nRF52840_D

K_User_Guide_v1.3.pdf.
[28] U. Guide, "Nordic Thingy : 91," 2019. https://infocenter.nordicsemi.com/pdf/Thingy91_UG_v1.0.pdf.
[29] D. S. Coombs, "OpenMV-H7," *Victorian Literature and Culture*, 2018. https://cdn.sparkfun.com/assets/a/3/f/d/9/OpenMV-H7_Datasheet.pdf.
[30] O. Audio, "Arducam Pico4ML Specifications Bluetooth Specifications Arducam Pico4ML Pinout." https://www.arducam.com/product/arducam-pico4ml-tinyml-dev-kit-rp2040-board-w-qvga-camera-lcd-screen-onboard-audio-b0330/.
[31] Rs-Components, "Datasheet Raspberry Pi Model B," *Raspberrypi.Org*, 2019. https://datasheets.raspberrypi.org.
[32] G. Monfort, "TinyML: From Basic to Advanced Applications," 2021. https://upcommons.upc.edu/bitstream/handle/2117/353756/160036.pdf.
[33] TensorFlow, "TensorFlow Lite for Microcontrollers," *TensorFlow*, 2021. https://www.tensorflow.org/lite/microcontrollers.
[34] "Edge Impulse," *Edge Impulse*. https://www.edgeimpulse.com/.
[35] "Embedded learning library." https://microsoft.github.io/ELL/.
[36] "ARM NN." https://www.arm.com/.
[37] "Pytorch mobile." https://pytorch.org/mobile/home/.
[38] "utensor.pdf." https://utensor.github.io/website/.
[39] Cartesiam, "NanoEdge AI Studio," 2020. https://cartesiam.ai/product/.
[40] "STM32Cube AI." https://www.st.com/content/st_com/en/ecosystems/artificial-intelligence-ecosystem-stm32.html.
[41] Y. S. Chong, W. L. Goh, V. P. Nambiar, and A. T. Do, "A 2.5 μW KWS Engine with Pruned LSTM and Embedded MFCC for IoT Applications," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 69, no. 3, pp. 1662–1666, 2021, doi: 10.1109/TCSII.2021.3113259.
[42] J. Kim, S. Chang, and N. Kwak, "PQK: Model compression via pruning, quantization, and knowledge distillation," *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, vol. 3, no. 1, pp. 1863–1867, 2021, doi: 10.21437/Interspeech.2021-248.
[43] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, [Online]. Available: http://arxiv.org/abs/1806.08342.
[44] C. C. Chung, W. T. Chen, and Y. C. Chang, "Using Quantization-Aware Training Technique with Post-Training Fine-Tuning Quantization to Implement a MobileNet Hardware Accelerator," *Indo - Taiwan 2nd Int. Conf. Comput. Anal. Networks, Indo-Taiwan ICAN 2020 - Proc.*, pp. 28–32, 2020, doi: 10.1109/Indo-TaiwanICAN48429.2020.9181327.
[45] S. Zhuo *et al.*, *An Empirical Study of Low Precision Quantization for TinyML*, vol. 1, no. 1. Association for Computing Machinery, 2022.
[46] E. Yvinec, A. Dapogny, and K. Bailly, "Gradient-Based Post-Training Quantization: Challenging the Status Quo," 2023, [Online]. Available: http://arxiv.org/abs/2308.07662.
[47] Y. Zhou, S. M. Moosavi-Dezfooli, N. M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," *32nd AAAI Conf. Artif. Intell. AAAI 2018*, pp. 4596–4604, 2018, doi: 10.1609/aaai.v32i1.11623.
[48] M. Kimhi, T. Rozen, T. Kopetz, O. Sirkin, A. Mendelson, and C. Baskin, "FBM: Fast-Bit Allocation for Mixed-Precision Quantization," pp. 1–23, 2022, [Online]. Available: http://arxiv.org/abs/2205.15437.
[49] S. Huang, H. Jiang, and S. Yu, "Hardware-aware Quantization/Mapping Strategies for Compute-in-Memory Accelerators," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 3, 2022, doi: 10.1145/3569940.
[50] S. Yang, Z. Gong, K. Ye, Y. Wei, Z. Huang, and Z. Huang, "EdgeRNN: A Compact Speech Recognition Network with Spatio-Temporal Features for Edge Computing," *IEEE Access*, vol. 8, pp. 81468–81478, 2020, doi: 10.1109/ACCESS.2020.2990974.
[51] A. Scenes, "KEYWORD SPOTTING SYSTEM AND EVALUATION OF PRUNING AND QUANTIZATION METHODS ON LOW-POWER EDGE MICROCONTROLLERS Jingyi Wang , Shengchen Li School of Advanced Technology , No . 111 Ren ' ai Road," no. November, pp. 3–7, 2022.
[52] A. Wong, M. Famouri, and M. J. Shafiee, "AttendNets: Tiny Deep Image Recognition Neural Networks for the Edge via Visual Attention Condensers," pp. 1–9, 2020, [Online]. Available: http://arxiv.org/abs/2009.14385.
[53] P. Bao, A. I. Maqueda, C. R. Del-Blanco, and N. Garciá, "Tiny hand gesture recognition without localization via a deep convolutional network," *IEEE Trans. Consum. Electron.*, vol. 63, no. 3, pp. 251–257, 2017, doi: 10.1109/TCE.2017.014971.
[54] A. J. Paul, P. Mohan, and S. Sehgal, "Rethinking Generalization in American Sign Language Prediction for Edge Devices with Extremely Low Memory Footprint," *2020 IEEE Recent Adv. Intell. Comput. Syst. RAICS 2020*, pp. 147–152, 2020, doi: 10.1109/RAICS51191.2020.9332480.
[55] Tecperson, "Sign Language MNIST," p. 7172, 2017, [Online]. Available: https://www.kaggle.com/datasets/datamunge/sign-language-mnist.
[56] D. Rasband, "ASL Alphabet Test," 2018. https://www.kaggle.com/datasets/danrasband/asl-alphabet-test.
[57] S. Fenanir, F. Semchedine, and A. Baadache, "A Machine Learning-Based Lightweight Intrusion Detection System for the Internet of Things," vol. 33, no. 3, pp. 203–211, 2019.
[58] L. Chen *et al.*, "Deep Neural Network Based Vehicle and Pedestrian Detection for Autonomous Driving : A Survey," vol. 22, no. 6, pp. 3234–3246, 2021.