Journal of King Saud University – Computer and Information Sciences 34 (2022) 1595–1623

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

A review on TinyML: State-of-the-art and prospects

Partha Pratim Ray

Department of Computer Applications, Sikkim University, India

ARTICLE INFO

Article history:
Received 31 October 2021
Revised 25 November 2021
Accepted 25 November 2021
Available online 30 November 2021

Keywords:
TinyML
IoT
Edge intelligence
Energy efficient AI
Resource constrained intelligence
Embedded AI

ABSTRACT

Machine learning has become an indispensable part of the existing technological domain. Edge computing and Internet of Things (IoT) together presents a new opportunity to imply machine learning techniques at the resource constrained embedded devices at the edge of the network. Conventional machine learning requires enormous amount of power to predict a scenario. Embedded machine learning – TinyML paradigm aims to shift such plethora from traditional high-end systems to low-end clients. Several challenges are paved while doing such transition such as, maintaining the accuracy of learning models, provide train-to-deploy facility in resource frugal tiny edge devices, optimizing processing capacity, and improving reliability. In this paper, we present an intuitive review about such possibilities for TinyML. We firstly, present background of TinyML. Secondly, we list the tool sets for supporting TinyML. Thirdly, we present key enablers for improvement of TinyML systems. Fourthly, we present state-of-the-art about frameworks for TinyML. Finally, we identify key challenges and prescribe a future roadmap for mitigating several research issues of TinyML.

Contents

1. Introduction . . . 1596
2. Background of TinyML . . . 1597
   2.1. Basics of TinyML . . . 1597
   2.2. Constraints of TinyML . . . 1598
   2.3. Definition of TinyML . . . 1598
   2.4. TinyML in edge . . . 1598
3. TinyML tool sets . . . 1600
   3.1. Hardware . . . 1600
   3.2. Software and libraries . . . 1600
4. Key enablers of TinyML . . . 1602
   4.1. TinyML-as-a-Service . . . 1602
   4.2. Hyperdimensionl computing . . . 1602
   4.3. Swapping . . . 1604
   4.4. Attention condensers . . . 1604
   4.5. Constrained neural architecture search . . . 1605
   4.6. Model compression . . . 1606
   4.7. Quantization . . . 1606
   4.8. Once-for-all network . . . 1608
   4.9. TinyML benchmark . . . 1609

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

E-mail address: ppray@ieee.org

https://doi.org/10.1016/j.jksuci.2021.11.019
1319-1578/

4.10. On-Device computing and accelerator . . . 1610
4.11. In-Processor learning . . . 1612
5. TinyML frameworks . . . 1613
5.1. TinyML frameworks by INCs . . . 1613
5.2. TinyML frameworks by research groups . . . 1614
5.3. Use cases in TinyMl . . . 1616
5.4. Speech recognition . . . 1616
5.5. Image recognition . . . 1616
5.6. Sign language prediction . . . 1616
5.7. Hand gesture recognition . . . 1617
5.8. Body pose estimation . . . 1617
5.9. Few-Shot keyword spotting . . . 1617
5.10. Always-on-voice wake up . . . 1617
5.11. Face detection . . . 1618
5.12. Cough related respiratory symptoms detection . . . 1618
5.13. Phenomics and ecological conservation . . . 1618
5.14. Autonomous vehicle . . . 1618
5.15. Anomaly detection . . . 1619
6. Prospects . . . 1619
6.1. Open challenges . . . 1619
6.2. Future road map . . . 1620
7. Conclusion . . . 1621
Declaration of Competing Interest . . . 1621
References . . . 1621

1. Introduction

Edge computing brings computation and data storage closer to the origin of data (Muniswamaiah et al., 2021). Edge computing provides an infrastructure to allow distributed computing play location sensitive acts (Anusuya et al., 2021) (Ying et al., 2021). The major advantage of edge computing is to provide low-latency and high-availability of several network aware services. Edge computing provides better privacy, security, and reliability to the network end-users (Bao et al., 2021). It leverages analytical computation resources very close to the end-users, resulting in higher throughput and better responsiveness in the applications (Lu and Lin, 2021). Several use cases can be considered as follows, smart healthcare, autonomous driving, public safety, human–machine interaction, agriculture, and emergency applications. Major benefit of edge computing is the reduction of the network traffic. Doing so, can help to run complex games and virtual reality aware events at the lightweight clients (Nezami et al., 2021) (Alwarafy et al., 2021).

Edge computing has tremendous potential to improve the automated network services with less burden on the network backhaul (Ding et al., 2021). IoT is such an instantiation of edge computing that allows billions of devices to transmit and receive the sensor originated data. IoT devices are deployed at the edge of the network with very-low processing capacity and memory footprint (Muhammad and Hossain, 2021) (Goudarzi et al., 2021). Majority of the edge devices that are integrated with IoT-based ecosystems are initially designed to collect sensor data and transmission of the data to neighborhood or remote cloud (Liu et al., 2021). IoT can help to move the computation away from the cloud to the edge of the network with a collaborative approach from sensors, edge devices, and cloud facilities (Singh et al., 2021). Such an orientation can provide data persistence, content caching, better service delivery, and quality IoT data management. Privacy and security can be significantly improved in this context (Wu et al., 2021). It can happen by shifting the security schemes getting shifted from cloud to IoT-edge devices. As an IoT-edge facility highly depends on the edge platforms for data collection and end-to-end data propagation, it minimally depends on the data transceiving through the long backhaul (Li et al., 2021b). However, it is a fact that such edge hardware is very resource constrained in nature that limits them to select high-end and complex services.

It is evident that currently edge computing can't solve everything, though it is expected to cater in near future (Guleria et al., 2021). One reason can be the huge difference between the hardware and web-based technologies that paves heterogeneous behavior. For example, machine learning applications require resource-full infrastructure to train, weight update, and deployment of the models (Ren et al., 2021b). Present scenario of IoT-edge is getting significant importance due to the need of deploy-ability of machine learning for smart use case development. Embedded system architectures are platform dependent that also hinders development of a standard machine learning framework for all IoT-edge systems. Further, the present technology domain presents a gap between machine learning dedicated embedded hardware and the required software to polish it (Ogino, 2021). Most of the existing embedded processors allow generic sensor data processing and web-based applications. Machine learning tool sets depend on the sophisticated hardware chips such as, graphics processing units (GPUs) and new dedicated hardware forms such as application specific integrated circuits (ASICs). These chips need enormous amounts of power and memory capacity to run deep neural network models. Present scenario depicts an undermining practice against the envisaged "*cloud-to-embedded*" aspect. Signal processing is also key for embedded intelligence, a paradigm to shift cloud intelligence to edge device – tiny embedded device for performing machine learning (ML) i.e., TinyML (Warden and Situnayake, 2019) (TinyML, 2021a,b).

The TinyML paradigm is still in its nascent stage that requires proper alignments for getting accommodated with existing edge-IoT frameworks. Pioneering research shows that the TinyML approach is crucial for smart IoT application development. But at the same time, several research questions (e.g., What is the need of TinyML? Is TinyML capable of running deep neural networks at the edge? How to keep the energy consumption less? Is high accuracy achievable by TinyML?) are identified that can hinder the growth of TinyML. In this paper, we discuss the background of the existing scenario behind TinyML. We also present a state-of-the-art review of literature that aims to cater to the significant usefulness of Tiny MLs numerous applications. Major contributions of this paper can be summarized as follows:

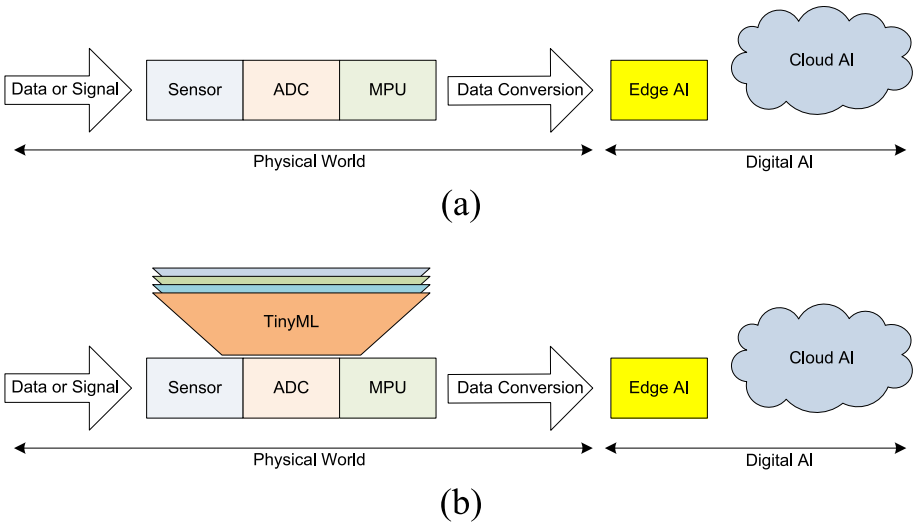

Fig. 1. (a) Existing physical world and digital AI, (b) TinyML assisted physical world and digital AI.

- To present intuitive understanding about the TinyML and provide detailed insight about the fundamentals thereto
- To present existing TinyML aware tool sets for model training and deployment at the edge where existing libraries, software packages, and hardware platforms are elaborated
- To discuss key enablers of TinyML paradigm to incorporate the concept of TinyML-as-a-Service, hyperdimensional computing, swapping, attention condensers, constrained neural architecture searching, model compression, quantization, one-for-all network, TinyML benchmark, on-device computing cum accelerator, and in-processor learning.
- To present state-of-the-art frameworks for TinyML wherein we discuss about TinyML framework by current companies and research groups.
- To illustrate use cases for TinyML where we its usage in speech recognition, image recognition, sign language prediction, hand gesture recognition, body pose estimation, few-shot keyword spotting, always-on-voice wake up, face detection, cough related respiratory symptom detection, phenomics and ecological conservation, autonomous vehicle, and anomaly detection.
- To identify key challenges and prescribe future road map for TinyML research where we discuss about various issues related to low-power computation, limited memory usage, hardware-software heterogeneity, lack of suitable benchmarking tool sets, lack of datasets, lack of popularly accepted models, edge computing infrastructures, edge platform orchestration, data and network management, software development for edge, and need of new machine learning models. We also present a future road map with help of few important steps that should be incorporated in future that includes edge intelligence framework, task offloading, mobility support, and level rating.

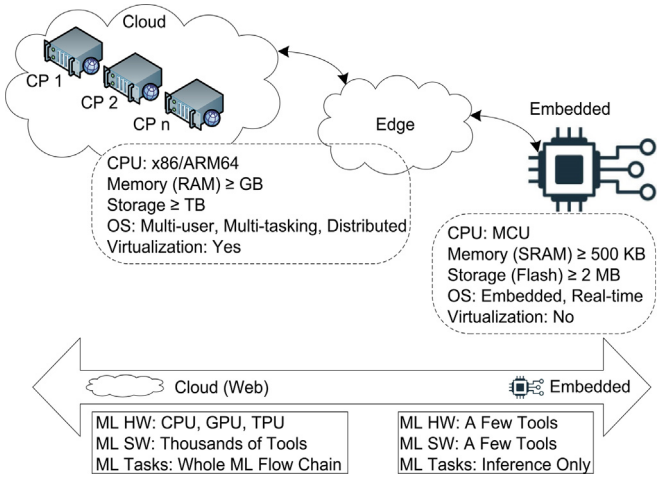

Fig. 2. TinyML has miniscule scalability.

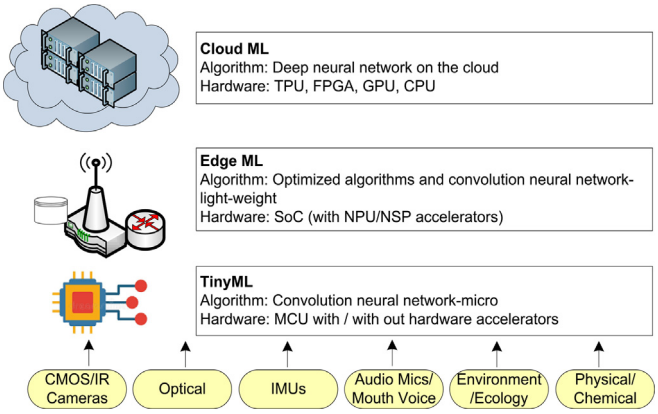

Fig. 3. Layered approach with respect to TinyML.

Rest of the paper is organized as follows. Section II presents the background of TinyML. Section III discusses key enablers of TinyML, Section IV presents state-of-the-art illustrations of frameworks for TinyML. Section V deals with some use cases involving TinyML. Section VII depicts key challenges and prescribes future road maps. Section VIII concludes the paper.

2. Background of TinyML

2.1. Basics of TinyML

TinyML is a paradigm that facilitates running machine learning at the embedded edge devices having very less processor and memory (ARM-TinyL, 2021) (Forbes-TinyML, 2021). The power consumption for such systems running machine learning should be within a few milliwatt or less. Typically, TinyML allows IoT-based embedded edge devices to go to lower power systems with amalgamation of sophisticated power management modules. Such a system should exploit the hardware acceleration (Learning,

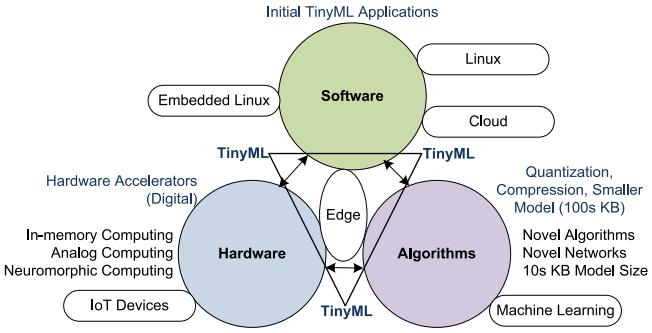

Fig. 4. Composition of TinyML.

2021). Moreover, the software that helps to run machine learning in the TinyML scenario should be as compact as possible so that power savings can be done. TinyML systems should specialize in optimizing various machine learning models to provide better accuracy under resource frugal constraints. TinyML system must accommodate following requirements, (i) energy-harvesting edge devices for running learning models, (ii) enables battery operated embedded edge devices, (iii) scalability to trillions of sensors enabled cheap embedded devices, and (iv) codes that can be stored within few KB in the on-device RAM (Recent Progress on TinyML Technologies and Opportunities, 2021) (Data Collection Design for Real World TinyML, 2021). Today's machine learning devices are hosted in public clouds as well as private premises. Organizations use ready-to-go deployed models from various learning aware cloud services in many industrial applications. Dependency on such cloud-based machine learning services paves few challenges such as, (i) huge energy consumption, (ii) privacy issues, (iii) network and processing latency, and (iv) reliability issues. Existing physical world takes raw data or signals from sensors and processes at the microprocessor unit (MPU). MPU helps to cater AI-aware analytics support with the help of specialized edge-aware AI systems. The edge AI can communicate with remote cloud AI for knowledge transfer. TinyML is aware that the physical world is smarter than the existing scenario (EdgeML: Algorithms for TinyML, 2021). Such systems can take decisions at the embedded edge devices before seeking help from edge AI or cloud AI. This setting results in the following improvements, (i) energy efficiency, (ii) better privacy of local data, (iii) low processing latency, and (iv) minimal connectivity dependency.Fig. 1

2.2. Constraints of TinyML

Major constraints that are currently hindering the growth of the envisaged TinyML paradigm has four key aspects, (i) energy: existing IoT-based embedded edge devices require minimum 10–100 mAh battery for stand-alone processing; thus efficient energy harvesting techniques should be deployed to power such edge devices to consume necessary energy for machine learning tasks, (ii) processor capacity: majority of tiny edge devices have 10–1000 MHz clock speed; it can restrict the complex learning models from running efficiently at the edge, (iii) memory: existing tiny edge platforms possess on average less than 1 MB on-board flash memory with 1000 KB SRAM; lack of space hinders the models to accommodate with the MCU, and (iv) cost: though individual device cost is low, a cumulatively higher scale can incur huge overall cost for massive deployment. Eradication of such issues are must for TinyML to succeed in low-cost edge platforms (Artificial Neural Networks, 2021) (MLOps for TinyML, 2021). Fig. 2 presents the comparison between TinyML with edge ML and cloud ML in terms of algorithm, hardware, and scalability (TinyML, 2021a). TinyML systems can take direct data input from various sensors. It can use a micro-nano level convolution neural network. The system can accommodate a microcontroller unit (MCU) with or without hardware accelerators. Edge-based ML devices can have optimized light-weight convolution neural networks to run on the system-on-chip (SoC) with a neural processing unit (NPU) with in-built accelerators. The process gets bulkier and highly computationally intensive at the cloud level where complex deep neural networks can be executed with help of GPU, multi-core CPUs, and tensor processing unit (TPU). Fig. 3. presents the layered approach of TinyML.

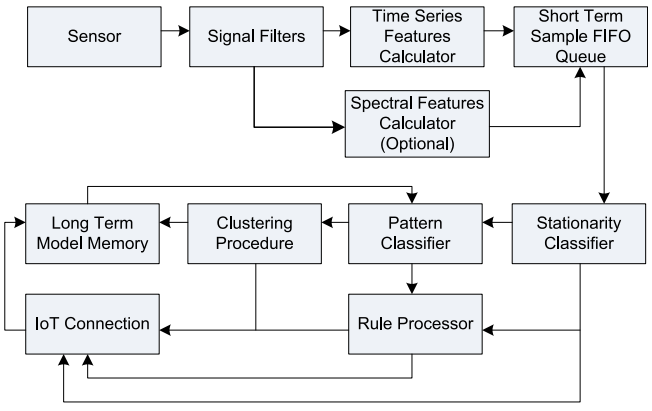

Fig. 5. Edge pipeline for TinyML for time series data.

2.3. Definition of TinyML

In this context, we can define TinyML as follows: "machine learning aware architectures, frameworks, techniques, tools, and approaches which are capable of performing on-device analytics for a variety of sensing modalities (vision, audio, speech, motion, chemical, physical, textual, cognitive) at mW (or below) power range setting, while targeting predominately battery-operated embedded edge devices suitable for implementation at large scale use cases preferable in the IoT or wireless sensor network domain" (TinyML, 2021a). Thus, TinyML can be envisaged as the composition of three key elements (i) software, (ii) hardware, and (iii) algorithms. TinyML can be accommodated in Linux, embedded Linux, and cloud-based software where initial TinyML applications can be run. The hardware can comprise IoT devices with or without hardware accelerators. Such devices can be based on in-memory computing, analog computing, and neuromorphic computing for better learning experience. Algorithms for the TinyML system should be novel so that KB sized models can be deployed in the resource frugal edge devices. Better compression and quantization schemes are evitable in this context (Endpoint AI and the Advent of the microNPU, 2021). Fig. 4. presents the essential components of TinyML where an optimal amalgamation of hardware-software co-design is a very important aspect. Such systems should overlap the orientations of optimized machine learning with high quality data and compact software design (Privacy in Context, 2021; Neural, 2021). Ordinarily, the TinyML system is flashed with binary files which are generated from the trained model on a larger host machine (Amber: A Complete, ML-Based, Anomaly Detection Pipeline for Microcontrollers, 2021).

2.4. TinyML in edge

Standard edge pipeline for TinyML setting is presented in Fig. 5 (Unsupervised collaborative learning technology at the Edge, 2021). The edge pipeline activity starts with sensors which collect raw data and provide the signal filters. The signal filters then filters the data based on the features dimension. For instance, if the data is in time series orientation, then time series features are computed. Optionally, spectral features may be computed. Samples

Table 1
Comparison Among Hardware Platforms to Support TinyML.

Hardware	Processor	CPU Clock	Flash	SRAM	Power/ Voltage	Connectivity	Sensors/Connectors	Organization
Apollo3	32-bit ARM Cortex-M4F	48 MHz, 96 MHz with TurboSPOT™	1 MB	384 KB	6μA/MHz Battery option,	BLE5, FTDI SPI, USB	Accelerometer, HM01B0 camera, MEMS microphone	SparkFun
STM32F Discovery	32-bit ARM Cortex-M4 FPU Core	48 MHz	1 MB	192 KB	3–5 V	LQFP100 I/O, USB	Accelerometer, microphone,	STMicroelectronics
ST IoTDiscovery	ARM Cortex-M4	48 MHz	1 MB,64Mbit Quad-SPI	128 KB	Battery option	8.211b/g/n, NFC, 868/915 MHz, BLE 4.1, USB	Microphone, accelerometer, gyroscope, barometer, gesture detection, humidity, temperature,	STMicroelectronics
ECM3532 AI Sensor NSP	ARM Cortex-M3, NXP CoolFlux 16-bit DSP	100 MHz	512 KB	256 KB	5μA/MHz, Battery option	BLE 4.2, RF, USB	Pressure, temperature, gyroscope, accelerometer, microphone	Eta Compute
Arduino Nano 33 BLE Sense	nRF52840	64 MHz	1 MB	256 KB	3.3 V, 15 mA/pin	UART, SPI, SPI, I2C, USB, BLE	IMU, microphone, gesture, light, proximity, barometer, temperature, humidity	Arduino
OpenMV Cam H7 Plus	ARM Cortex-M7	480 MHz	2 MB (Internal)	1 MB, 32 MB SDRAM	3.7 V Li-Ion	I2C, USB, CAN, UART,	5MP Camera at 50 FPS	OpenMV
Himax EW-I Plus	32-bit ARC EM9D DSP with FPU Core	400 MHz	2 MB	2 MB	1.2–3.3 V, Battery	SPI2, I2C, UART, USB	VGA Camera 60 FPS, accelerometer, microphone	SparkFun
Thunderboard Sense 2	EFR32™ Mighty Gecko Wireless SoC	38.4 MHz	1 KB	256 KB	3.3–5 V, Coin cell, ULP	2.4 GHz, USB, SPI,	Temperature, humidity, ambient light, pressure, air quality, microphone, hall-effect, UV	Silicon Labs
Sony's Spresense	ARM Cortex-M4F 6 Core	156 MHz	8 MB	1.5 MB	3.3–5 V	SPI, I2C, UART, I2S, GNSS antenna	Microphone, camera	Sony
TinyML Board	Syntiant® NDP101 NDP, 32-bit ARM Cortex-M0	48 MHz	256 KB	32 KB	3.7–5 V, LiPo battery	UART, I2C	Motion, microphone	Sytiant
Arduino Portenta H7	ARM Cortex-M7, ARM Cortex-M4 GPU	480 MHz, 240 MHz	16 MB	8 MB SDRAM	3.7–5 V, Li-Po cell, 700mAh	WiFi, BLE, 10/100 Ethernet Phy, USB, MIPI DSI, MPI D-PHY	Temperature, camera extension	Arduino
Raspberry Pi 4B	64-bit ARM Cortex-A72 quad core, Broadcom BCM2711	1.5 GHz	–	256 KB	3.8–4 W, 3.3–5 V	WiFo, BLE, CSI, DSI, HDMI, USB, Ethernet	Temperature	Raspberry Pi
AI-deck 1.1	GAP8, ESP32	168 MHz	1 MB	192 KB	3–5 V	WiFi, URT, SPI	Monochrome camera	Bitcraze
Pico4ML BLE	Raspberry Pi RP2040 DSP dual core	133 MHz	4 MB	264 KB	1.7–3.6 V, battery	BLE, USB, I2C	Camera QVGA 60 FPS, microphone, IMU	ArduoCam
MKR Video 4000	Intel® Cyclone ® 10CL016 FPGA, , 32-bit ARM Cortex M0	48–200 MHz	2 MB, 256 KB	32 KB, 8 MB SDRAM	3.7 V Li-Po, 1024mAh	SPI, I2C, UART, USB, MIPI, u-blox NINA-W102	–	Arduino
MKR Video 4000	Intel® Cyclone ® 10CL016 FPGA,, 32-bit ARM Cortex M0	48–200 MHz	2 MB, 256 KB	32 KB, 8 MB SDRAM	3.7 V Li-Po, 1024mAh	SPI, I2C, UART, USB, MIPI, u-blox NINA-W102	–	Arduino
Nicla Sense ME	ARM Cortex M4	64 MHz	512 KB	64 KB	3.7 V Li-Po	BLE4.2, SPI, USB, I2C	Accelerometer, gyroscope, pressure, geomagnetic, gas, temperature, humidity	Arduino

(*continued on next page*)

Table 1 (*continued*)

Hardware	Processor	CPU Clock	Flash	SRAM	Power/ Voltage	Connectivity	Sensors/Connectors	Organization
CC1352P Launchpad	CC1352R Wireless MCU LaunchPad™	48 MHz	352 KB	8 KB	60µA/ MHz ,1.8–3.8 V	UART, I2C, SSI, I2C, I2S, 868/915/ 433 MHz, BLE, Thread, ZigBee, 802.15.4, Sub-1 Ghz	Temperature	TI
Hardware	Processor	CPU Clock	Flash	SRAM	Power/ Voltage	Connectivity	Sensors/Connectors	Organization
ESP-EYE	32-bit ESP32	240 MHz	4 MB	8 MB PSRAM	3.3 V	WiFi, USB, SPI, I2C, UART, BLE	2MP camera	Espressif
GAP8	RISC-V, hardware convolution engine	250 MHz (FC), 175 MHz (C), 22.65GOPs	512 KB	80 KB,8 MB SDRAM	1.8–3.3 V, 4.24mW/ GOP	Serial, SPI, I2C, I2S, CPI, Hyperbus, UART	Extension camera	Green Wave Technologies
GAP9	RISC-V, hardware convolution engine	400 MHz, 150.8GOPs	1.5 MB	128 KB, 2 MB External	1.8–3.3 V, 0.33mW/ GOP	Serial, SPI, I2C, I2S, CPI, Hyperbus, UART	Extension camera	Green Wave Technologies
Nordic Semi nRF52840 DK	ARM Cortex M4	64 MHz	192 KB	24 KB	1.7–5 V Li-Po	BLE5, Bluetooth mesh, Thread, Zigbee, 802.15.4, ANT, 2.4 GHz, NFC, UART	–	Nordic
Nordic Semi Thingy:91	ARM Cortex M33, nRF9160 SiP	64 MHz	1 MB	256 KB	1440mAh Li-Po	UART, SPI, I2S, NB-IoT, LTE-M	Color, light, humidity, air quality, temperature, pressure	Nordic
XCore.ai	Convolution and dense neural network FPU 16 core	3200MIPS, 1 M 512 FFTs/s	–	1 MB	1.8–3.3 V, 500mW	UART, SPI, I2S, I2C, MIPI, USB	–	XMOS
FRDM-K64F	ARM Cortex M4	120Mhz	1 MB	256 KB	1.7–3.6 V, Coin cell	Ethernet, CAN, SPI, I2C, UART, I2S	Accelerometer, magnetometer	Mbed

are then kept inside a first-in-first-out (FIFO) data structure for a very short term. If the data is in time series format, then the stationarity classifier is used to check where the data follows stationary attributes. Next phase aims to pave IoT-based connection to long term model memory which in turn communicates with the pattern classifier for rule-based processing or cluster procedure, depending on the context of application. The edge pipeline can be modified as per the requirement of the cross-section data when needed.

3. TinyML tool sets

TinyML requires several hardware specifications, libraries, and software platforms to leverage predictions. We present a brief about existing hardware and software tool sets being investigated for possible TinyML deployment.

3.1. Hardware

We select several TinyML aware hardware platforms such as, Apollo3 (Apollo3, 2021), STM32F Discovery (STM32F, 2021), ST IoT Discovery (ST IoT Discovery, 2021), ECM3532 AI Sensor Neuro sensor processor (NSP) (ECM3532, 2021), Arduino Nano 33 BLE Sense (Arduino Nano 33, 2021), OpenMV Cam H7 Plus (OpenMV, 2021), Himax EW-I Plus (Himax, 2021), Thunderboard Sense 2 (Thunderboard Sense 2, 2021), Sony's Spresense TinyML Board (Sony's Spresense TinyML Board, 2021), Arduino Portenta H7 (Arduino Portenta H7, 2021), Raspberry Pi 4B (Raspberry Pi 4B, 2021), Nvidia Jetson Nano (Nvidia Jetson Nano, 2021), CC1352P Launchpad (CC1352P Launchpad, 2021), ESP-EYE (ESP-EYE, 2021), GAP8 (GAP8, 2021), GAP9 (GAP9, 2021), AI-deck 1.1 (AI-deck 1.1, 2021), Seeed Wio Terminal (Seeed Wio Terminal, 2021), Agora Product Development Kit (Agora Product Development Kit, 2021), Pico4ML BLE (Pico4ML BLE, 2021), MKR Video 4000 (MKR Video 4000, 2021), Nicla Sense ME (Nicla Sense ME, 2021), Nordic Semi nRF52840 DK (Nordic Semi nRF52840 DK, 2021), Nordic Semi Thingy:91 (Nordic Semi Thingy:91, 2021), XCore.ai (XCore.ai, 2021), and FRDM-K64F (FRDM-K64F, 2021). There are many other alternatives available in the current market which can also be investigated for suitability for TinyML application development. We present Table 1 to compare among the mentioned hardware platforms in terms of processor, CPU clock frequency, flash memory, SRAM size, power or voltage consumption, connectivity, sensors or connectors and product developer. We notice that majority hardware boards process below 100 MHz processor frequency with average less than 1 MB flash and less than 1 MB SRAM. Bluetooth (BLE) and Wi-Fi are mostly chosen connectivity technologies. We find that most of the boards facilitate a number of on-board sensors including accelerometer, temperature, humidity, microphone, gyroscope, air pressure, gesture detection, light sensor, hall-effect, air quality, and camera. Power consumption of such boards is around the mW range. Most of the devices can be operated from Li-Po and coin batteries besides regular DC power supply. We also notice that ARM Cortex-M4 is the most popular processor among all other alternatives. Few boards (e.g., GAP8, GAP9) are in-built with a hardware convolution engine (HCE) to enhance the neural network aware computation at the edge.

3.2. Software and libraries

- *TensorFlow Lite (TFL):* It is an open-source deep learning framework for supporting edge aware learning inference. Edge aware on-device machine learning can be addressed by this framework while leveraging five key constraints (e.g., latency, privacy, connectivity, size, and power consumption). It supports Android, iOS, embedded Linux, and a variety of microcontrollers

(TensorFlow Lite, 2021). It also supports languages (e.g., C++, Python, Java, Swift, Objective-C) to develop machine learning on the edge device. Model optimization with hardware acceleration is paved by TFL. A range of AI applications covering classification (e.g., image, text), question answering, object detection, and pose estimation can be easily supported. The size of its binary is ∼ 1 MB, given that all the operators are connected to 32-bit ARM builds. It can generate as low as 300 KB binary when using some operators for image classification. Whole work process in TFL follows by selecting 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 to 8-bit integers. TensorFlow Lite Micro (TSFM) is an extension of TFL that aims to run machine learning in KB size ARM Cortex processors. TFLM is written in C++ 11 and runs on 32-bit platform (e.g., ESP32, Arduino nano 33 BLE Sense, SparkFun Edge, STM32F746 Discovery Kit, Adafruit EdgeBadge, Bluefruit, ESP-EYE, ESP32-DevKitC, Wio Terminal, Himax WE-I, Sony Spresense, and Synopsys DesignWare ARC EM). However, it doesn't support on-device training.

- *uTensor:* It is a free embedded learning environment that helps to prototype and rapid deployment at the IoT-edge devices (uTensor, 2021). It includes an inference engine, a graph processing tool, and upcoming data collection architecture. It takes a neural network model by using Keras for training. It then converts the trained model into a C++. The uTensor helps to convert the model for suitable deployment in the Mbed, ST, and K64 boards. The uTensor is a small size module that requires only 2 KB on disk. A Python SDK is used to customize the uTensor from ground up. It depends on the following tool sets such as, Python, uTensor-CLI, Jupyter, Mbed-CLI, and ST-link (for ST boards). Initially, a model is created and then defined with a quantization effect. The next step is code generation for suitable edge devices.
- *Edge Impulse:* It is a cloud service for developing machine learning models in the TinyML targeted edge devices. This supports AutoML processing for edge platforms (Edge Impulse, 2021). It also supports a number of boards including smart phones to deploy learning models in such devices. Training is done on the cloud platform and the trained model can be exported to an edge device by following a data forwarder enabled path. The impulse can be run in local machine by the help from the in-built C++, Node.js, Python, and Go SDKs. Impulses are also deployable as a WebAssembly library.
- *NanoEdge AI Studio:* The software was earlier known as the cartesiam.ai, now enables selection of the best library and test library's performance by using an emulator before final deployment in the edge (NanoEdge AI Studio, 2021). It has many important features that include (i) limiting maximum flash memory requirement during project creation, (ii) frequency filtering, (iii0 flash memory optimization, (iv) serial data plotting, (v) real-time search, and (vi) selection of libraries after benchmarks. It can be used to detect anomalies in dataset and classification tasks. It supports STM32 Nucleo-32 board and Arduino Nano 33 IoT board
- PyTorch Mobile: It belongs to the PyTorch ecosystem that aims to support all phases starting from training to deployment of machine learning models to smart phones (e.g., Android, iOS). Several APIs are available to preprocess machine learning in mobile applications (PyTorch, 2021). It can support the scripting and tracing of TorchScript IR. Further support is given for the XNNPACK 8-bit quantized kernel targeting ARM CPUs. It can also support GPUs, digital signal processors, and neural processing units. Optimization facility for mobile phone deployment is paved via the mobile interpreter. Currently it supports image segmentation, object detection, video processing, speech recognition, and question answering tasks.
- *Embedded Learning Library (ELL):* Microsoft has developed the ELL for supporting TinyML ecosystem for embedded learning (ELL, 2021). It provides support for Raspberry Pi, Arduino, and micro:bit platforms. The models which are deployed in such devices are internet agnostic, thus no cloud access is required. It supports the image and audio classification at the moment.
- *STM32Cube.AI:* It is a code generation and optimization software that allows machine learning and AI related tasks easier for STM32 ARM Cortex M–based boards (STM32Cube.AI, 2021). Implementation of neural networks in STM32 board can be directly achieved by using STM32Cube.AI to convert the neural nets into an optimized code for most appropriate MCU. It can optimize the memory usage during run time. It can use any trained model by conventional tools such as TFL, ONNX, Matlab, and PyTorch. This tool is actually an extension of the original STM32CubeMX framework that helps STM32Cube.AI to perform code generation for target STM32 edge device and middleware parameter estimation.
- *µTVM:* MicroTVM is an extension of existing tensor virtual machines (TVM) to facilitate execution of tensor programs on microcontroller boards. It allows the optimization of these programs via the AutoTVM platform that helps to optimize tensor

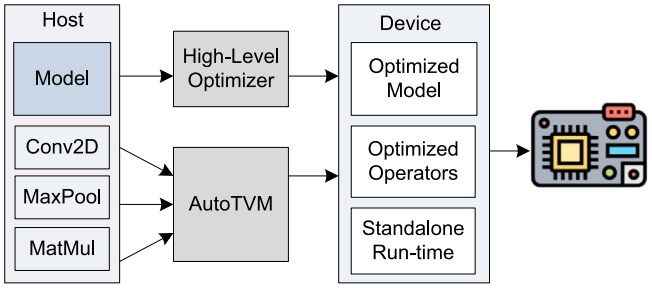

Fig. 6. System model for µTVM optimization and deployment to microcontroller.

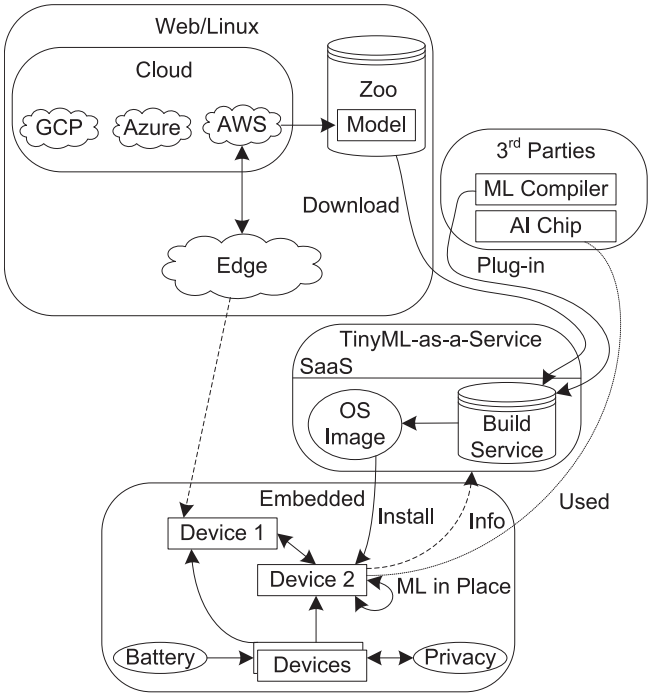

Fig. 7. TinyML-as-a-Service.

programs (uTVM, 2021). In practice, a microcontroller is first connected with the desktop or high-end machine that is running the TVM in the background via USB-JTAG port. Desktop runs the OpenOCD to provide the connection between the microcontroller and the desktop. Doing so, OpenOCD supports μTVM to control the microcontroller by applying a device-agnostic TCP port. User should provide particulars (e.g., C cross-compiler toolchain for microcontroller, method for read/write/execute on device's memory, specification about device's architectural layout, and code snippet for preparing the device to execute the function) for getting support from μTVM. The μTVM requires the MicroSession to have connection with the device based on the given method (e.g., OpenOCD). Later, the μTVM runtime is cross compiled as per the cross-compiler supplied earlier. Finally, the binary of the compiled code is loaded to the device. One can face various aspects of μTVM association with TinyML, for example, lazy execution, tensor loading, function calling, and module loading. Fig. 6. presents the system model consisting of μTVM for deploying optimized models to microcontrollers i.e., TinyML-based edge devices (uTVM system, 2021).

4. Key enablers of TinyML

4.1. TinyML-as-a-Service

TinyML-as-a-Service or TinyMaaS aims at solving some of the important problems related to machine learning for embedded domains such as the efficient business development process for ML in the IoT environment. Conventional cloud-based ML tasks are leveraged by a set of cloud providers (CP) which are equipped with a CPUs, GPUs, and tensor processing unit (TPU). On other hand, the embedded devices with minimal processing and memory capabilities are not deemed to be suitable to run full-fledged ML models (Doyu et al., 2020). The cloud-based web services provide thousands of various tool sets to process the whole ML flow chain starting from data collection, preprocessing, data transformation, model training, model deployment, and inference. Whereas, embedded devices are only fit for ML model inferences. Such a huge scale-wise gap makes the task of the embedded devices challenging for ML augmentation. Ordinarily, pre-trained ML models require huge computational and infrastructural resources that resource constrained IoT-based devices may not be capable to leverage of. Thus, such models should be optimized for size fitting well before loading such models to IoT devices. An ML compiler can translate the pre-trained ML models for deployment to a target IoT device. Minimization of models can be done by allowing one of the following techniques such as, *quantizing* (fewer bits for computation), *pruning* (eradicating useless parameters), and *fusing* (combining multiple operators together into one). TinyMLaaS ecosystem should require a number of ML compilers which can generate specialized light-weight ML runtime models for a given embedded platform. It is also worthy to inculcate ML models suitable for specific IoT-based hardware accelerators i.e., chip manufacturer dependent. The major focus of this service aspect is to provide customized on-demand facility to the product developers. One can think of using the Light-weight machine-to-machine (LwM2M) along with on-the-fly model inferencing modules (e.g., Zoo) for generation of appropriate ML model for IoT device. Such a holistic ecosystem can minimize the product development duration for the embedded designers who may wish to select variety of ML algorithms and models. Thus, one should expect a plausibly high impact on the forthcoming business process involving the embedded ML development. Fig. 7. presents the TinyMLaaS architecture.

TinyMLaaS can mitigate the privacy concern for business by keeping user data within the physical "on-premises" boundaries. It can try to confine the processing of business sensitive data only at the IoT device itself in the enterprise solutions. It can also help in reduction of the network bandwidth while focusing a set of IoT end-devices for performing heterogeneous tasks). TinyMLaaS should assume that the majority of the IoT devices are equipped with narrowband connectivity (e.g., NB-IoT) where a device can have very limited possession of data transmissions. Such indicates the importance of "on-premises" data augmentation resulting in the work of data to offloading at the IoT-edge. Further, one can envisage the ultra-reliable and low-latency aware services under the aegis of the TinyMLaaS where inference of ML models is possible at the device level. Moreover, the IoT devices that are deployed in a wide range of unstable network coverage areas (e.g., rural areas, sea, mountain) should be enabled with on-device decision making based on feature wise predictions (TinyML as-a-Service, 2021). Doing so, it will subsequently minimize the power consumption aspects and improve the energy efficiency. One can consider such last mile Io devices as battery powered that can perform processing locally with a minimal intervention of edge or cloud data connectivity.

4.2. Hyperdimensionl computing

Hyperdimensional computing (HDC) provides an alternative option to the existing learning techniques with lightweight algorithms. Such minimal algorithms consume very less energy as compared to conventional techniques. In this approach, all the data points are represented by high-dimensional vectors i.e., hyper vectors. The hypervectors are mapped to the high-dimensional space i.e., hyperspace for completion of the computational task. Ordinarily, a large hyper vector dimension ($D \geq 1000$) is required to achieve at par accuracy when compared to conventional neural learning techniques. However, an excessive increase of hypervector dimension can incur higher computational cost and hardware cost, resulting in undermined benefits. Table 2 presents the comparison between classical and HDC classification (Ge and Parhi, 2020).

In this context, one can expect TinyML to consider the hyper vectors as a promising component for leveraging a new horizon of embedded intelligence. HDC differs from neural learning algorithms in terms of primary data type where raw samples are mapped to the random high-dimensional vectors i.e., *sample hypervectors*. In the next phase, similar sample high-dimensional vectors are combined in linear fashion to come up with an ensemble class of hypervectors known as the *class encoders*. A *query hyper vector* (Q) is later generated during the inference process based on a given input as per the process mentioned for the sample hypervectors. At

Table 2
Comparison between classical and HDC classification (Ge and Parhi, 2020)

Computing Types	Classical Computing	HDC Computing
Data Type	Bit	Hypervector
Data Transmission	Addition, Multiplication, Logic	Add-Multiply-Permute
Storage	Memory	Item Memory, Associative Memory
Training	Weights	Class Hypervectors
Testing	Run Pre-trained Classifier	Associate Query Hypervectors with Class Hypervectors
Model Complexity	High	Low
Accuracy	Very High	Acceptable
Feature Encoding	Easy	Difficult
Number of Features	Many	One

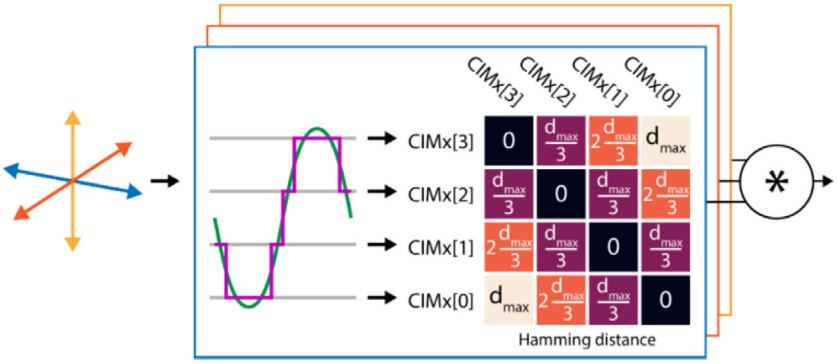

Fig. 8. HDC-based quantized signal representation from accelerometer sensor.

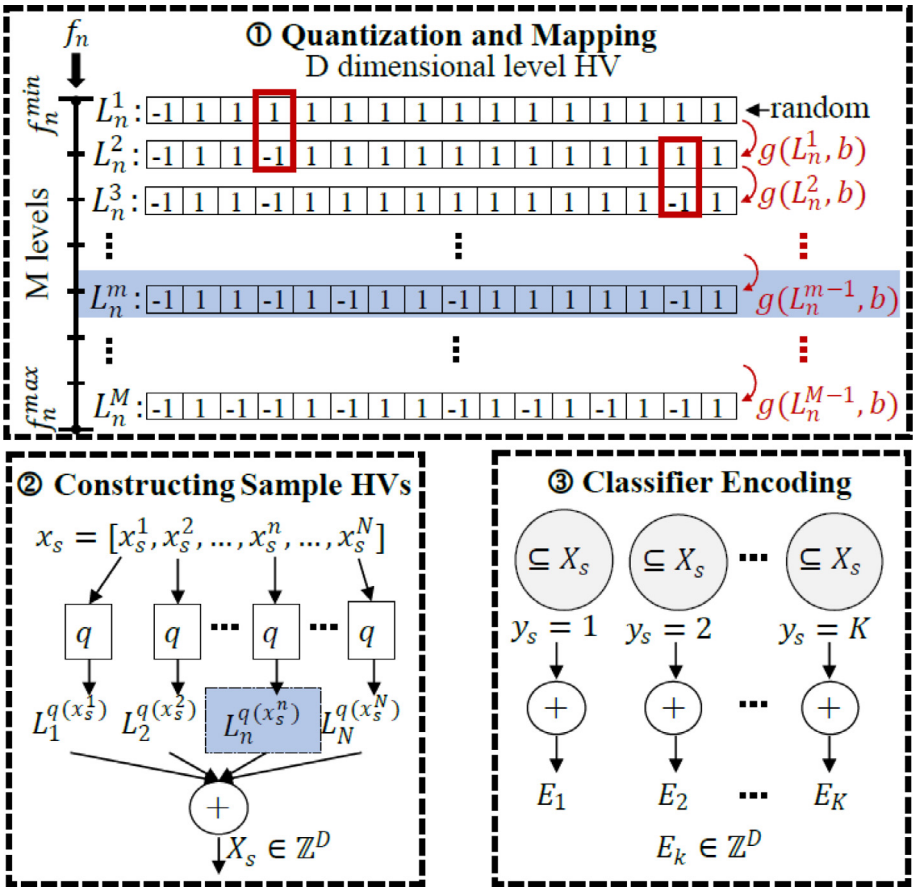

Fig. 9. HDC training phases with arbitrary bit values.

the last phase, the classifier tends to find the closest class hypervectors to Q based on the hamming distance or cosine similarity metrics. Despite high accuracy rates, high demand for memory and energy aware processing capabilities limits the hypervectors for the IoT-based embedded devices. Minimization of hyper vector dimension should be investigated to provide optimum accuracy rate against the enhanced robustness of the classifier. In (Zhou et al., 2021), limb-position aware hand gesture recognition system is proposed by using the HDC technique. This work shows context-aware orthogonalization for classifying gestures in multiple limb positions. To achieve this, firstly the electromyogram (EMG) signal is projected into the hypervectors and classified according to the baseline HDC classifier. In the next stage, a dual-stage architecture is imposed over the classification to emulate the context-based orthogonalization. The work ends with the direct encoding of accelerometer sensor features into the context of the hypervectors. Fig. 8. Presents HDC-based quantized signal representation from accelerometer sensor.

In (Basaklar et al., 2021), a mechanism is presented to minimize the dimensions of the HDC classifier to reduce memory and power consumption to align its orientation as per the IoT devices. The HDC training phase mentioned in this study comprises three steps such as, (1) quantization and mapping, (2) construction of sample hypervectors, and (3) classifier encoding as shown in Fig. 9. which is also known as the *baseline HDC implementation*. In the quantization and mapping step, the input space is quantized in D-dimensional *level hypervectors*. A low dimension quantization and high dimensional mapping are subsequently performed. In the sec-

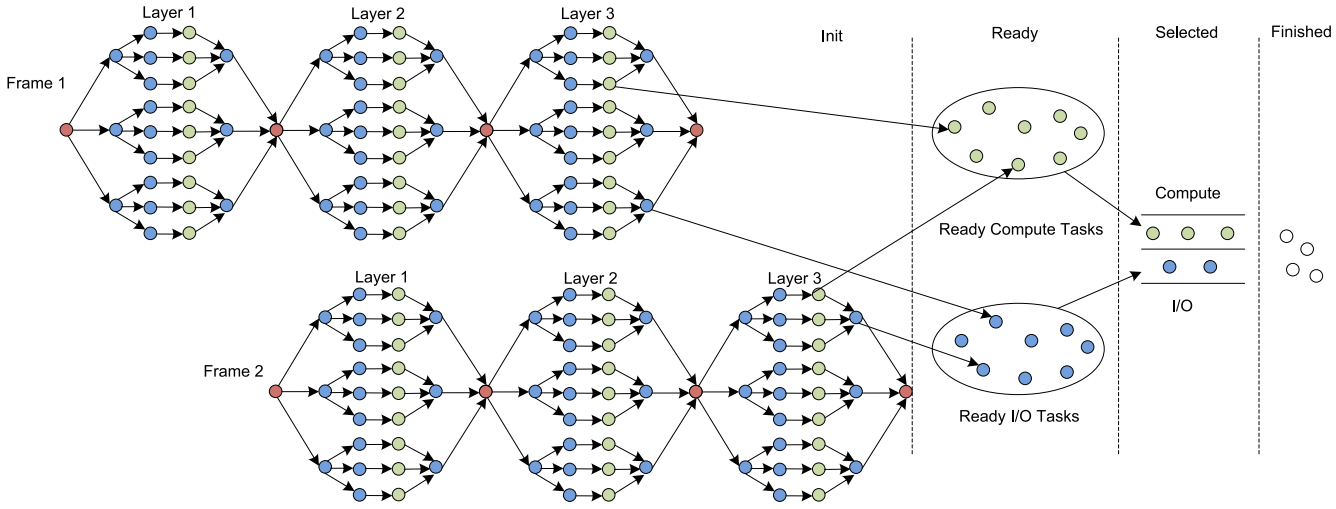

Fig. 10. SwapNN architecture.

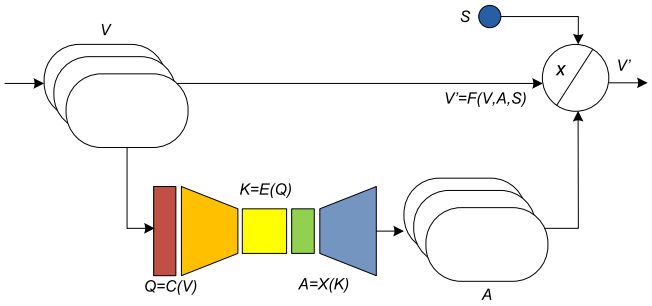

Fig. 11. Attention condenser with a condensation layer.

ond step, construction of sample hypervectors is done by using the input sample and level hypervectors. Finally, the classifier encoding is performed by adding all sample hypervectors with predefined labels. The work optimizes the trade-off between the classifier accuracy and the robustness with more than twice of the regular robustness.

4.3. Swapping

Existing neural network models need higher processing ability which is surely a big issue for IoT-based devices having very less static random-access memory (SRAM) of the microcontroller unit. Numerous techniques are investigated to deploy neural networks in the IoT ecosystem, however most of them sacrifice accuracy and generality in doing so. In (Miao and Lin, 2021) a new method is presented to execute neural networks in IoT centric microcontrollers by using *swapping*. In this approach neural networks are swapped between the tiny microcontroller's memory and the external large flash memory. The out-of-core neural network allows splitting one neural network layer's working into multiple series of tiles. Each of these tiles can be loaded into a tiny memory space of an IoT device. Upon a demand is raised, a requested chunk of the neural network layer is swapped from the external flash or memory SD card to the main memory. Excessive swapping may cause the micro-SD card loss durability along with execution slowdown of input/output operations, lack of security, and increase of energy consumption. The paved method demonstrates several acts to overcome such challenges. For example, hiding swapping delays with improved parallelism at minute granular levels. Fig. 10. presents the SwapNN architecture covering scheduling of input/output tasks in the tiles, layers, and frames parallelly. The process starts with extraction of CPU/IO parallelism to hide input/output delays. Firstly, it performs parallelism in the neural network layer to produce *Tile0*. Microcontroller can then use this *Tile0* to compute the next *Tile1* while swapping the tiles between the on-board memory and the external flash. Layer parallelism is the next option where input/output tasks are executed simultaneously. At the last phase, pipeline parallelism is achieved across the data frames. The benefit of this method is to compute both memory and IO bound layers in parallel for different frames. Two major types of tasks can be performed in the SwapNN architecture namely compute task (computer output tiles given that input tiles are available) and IO task (performs read/write of tiles from/to the external SD or flash memory). The task state of SwapNN defines the life cycle for each of the tasks related to IO and computation. For example, (i) INIT: used to set initialize the state during the creation of building graph, (ii) READY: when all predecessors are done with their job, a task becomes ready while keeping the in-degree counter reach 0, (iii) SELECTED: when memory is allocated to a task it switches from READY to SELECTED state, and (iv) FINISHED: upon completion of an IO task, it is switched to FINISHED state; at this point the in-degree counter is incremented by 1 so that all the successors can free-up the memory buffers.

This study shows that dong SwapNN has very less impact on the external SD card in terms of durability loss and lifetime. SwapNN adds more energy efficiency to the IoT device by minimizing overall power drop for the microcontroller unit. It is noticed that with a sufficiently large buffer or tile size, SwapNN sees a very throughput loss. Due to the parallel execution, overall IO overhead is reduced. Moreover, the large tile size allows the IO-bound layers to increase delay. This swapping neural networks can be considered as a promising technique for inclusion in the TinyML genre.

4.4. Attention condensers

Attention condensers are introduced as the key enabler of deep neural networks at the IoT-edge devices where low memory and processing capabilities are present. It can be used for a multitude of applications that includes complex speech and image recognition at the edge devices. An attention condenser can be considered as the self-attention mechanism that can self-learn and produce a condensed embedding. Such an embedding can characterize the joint local as well as cross-channel activation relationships. It allows us to perform selective attention as required. Attention con-

densers are different in terms of generic self-attention techniques which are designed to support deep convolutional neural networks. The key difference lies in the holistic augmentation of self-contained and stand-alone modules that can facilitate the larger sparser towards more frequent usage of attention condensers. Fig. 11. presents a design of the attention condenser having a condensation layer – C(V). The condensation layer is chained together with an embedding structure – E(Q). The whole design involves an expansion layer – X(K) and a selective attention approach – F(V,A,S). The task of C(V) is to help condense the input activation V to reduce the dimensionality to the condenser. Such dimension minimization is fed to Q to emphasize several activations that are placed very close proximity to the strong activations. The E(Q) then learns a condensed embedding (K) and produces such K from Q for characterization joint local along with cross-channel activation duo. Next phase aims to produce selective attention F(V,A,S) upon generation of self-activation values A from X(K). Such a deed is important to increase the dimensionality so that an output V' can be produced as a function of input activations (V), self-attention values A, and scale S.

Attention condensers can open up a new dimension between embedded intelligence and machine learning for a multitude of applications. One can expect attention condensers to facilitate tetherless machine learning in the extreme edge of the IoT ecosystem. It can enhance real-time decision-making capability at the IoT devices while preserving privacy, security, and dependability. The focus of attention condensers is to minimize the computational resources at the embedded devices. Its usage can be extended to the high-efficiency embedded deep neural networks for provisioning a variety of tasks such as drug discovery, natural language processing and visual perception.

The AttendNets (Wong et al., 2020b) is such a recently introduced deep neural network which is highly compact and designed for deployment at the extremely resource constrained IoT devices for image recognition applications. The AttendNets depend on the philosophy as mentioned earlier to extend stand-alone attention condensers for enhanced spatial-channel selective attention mechanisms. A machine design exploration scheme is employed on AttendNets to formulate both macro and micro architectural aspects of machine-driven designs. It shows promising results when compared to ImageNet$_{50}$ benchmark dataset in terms of accuracy, parameter reduction, minimization of memory utilization, and lowering multiply–add operations.

Another recent work demonstrates an attention condenser neural network that can achieve the semantic segmentation at the IoT edge device i.e., AttendSeg. This semantic segmentation scheme aims at device level low-precision but high compact deep neural network deployment.

4.5. Constrained neural architecture search

Present scenario requires deep neural networks to get deployed in the resource constrained IoT devices. However, there exists several research challenges for finding appropriate techniques towards the neural architecture search (NAS). Ordinary NAS comprises three steps, (i) search space, (ii) search algorithm, and (iii) the evaluator. The search space refers to the number of different architectures suitable for different hyperparameters. The search algorithm explores the available search space to look for the best possible architecture that aims at the maximization of the objective function. Finally, the evaluator can be used to find the model's accuracy against the efficiency of the deployed model. The NAS mechanism should involve the constraints (e.g., memory, processing delay) of the embedded devices as the objective function so that search space can be optimized for selection of appropriate deep neural networks. The plausible NAS for constrained hardware

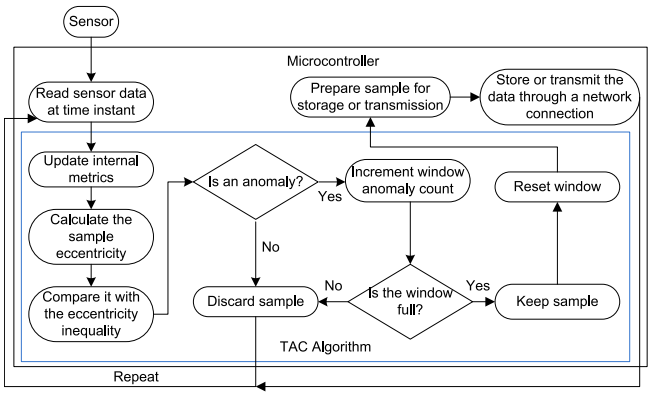

Fig. 12. Date flow in TAC algorithm.

domain should possess many objectives and several views related to the platform's capability to run neural networks (Benmeziane et al., 2021). The goals for such a NAS strategy can be classified into three types, (i) single target fixed configuration, (ii) single target multiple configurations, and (iii) multiple targets. Majority of the existing hardware platforms fall under the category of single target multiple platform The NAS generally tries to achieve the best hardware architecture based on its accuracy that follows one of the two options e.g., hardware-aware search strategy (search algorithm finds the accuracy metrics from the accuracy evaluator based on energy consumption memory usage, and latency) and hardware-aware search space (rule based restricted pool of hardware). The single target multiple configurations approach selects the hardware that provides optimal architecture based on accuracy and latency profile. The multiple targets strategy finds the best architecture when a set of hardware platforms is provided to optimize accuracy metrics. This scheme seems to be the most challenging due to the fact that targeting multiple hardware platforms may contain intersection of all the architectures.

The constrained NAS can be assumed as the cast as a multi-objective optimization problem that considers both accuracy and hardware constraints. One can use a two-stage optimization technique under the single-objective optimization problem to find accuracy aware hardware platforms. Neural networks that fall under the reinforcement learning and evolutionary computing may be searched against the constrained optimization aware NAS problem. On the other hand, parametrized aggregation aspects can be sought for scalarization to transform multi-objective optimization problems. The NAS should involve the following hardware search space for selection of the optimal method of neural network. For example, server processors (CPUs, GPUs, field programmable gate arrays - FPGAs, and ASICs), mobile devices, tiny devices, parameter-based (suitable for FPGAs), and template-based (suitable for ASICs).

The hardware selection should also cater related metrics to decide which platform should be used for a given neural network. For example, the latency of processors at inference time, energy consumption, form factor (area), floating point operations per second (FLOPs) memory footprint (real-time use, lookup table models, analytical estimation, and prediction models). A recent work presents the μNAS architecture to deploy a neural network into a KB-sized RAM for the IoT-based microcontrollers (Liberis et al., 2021). Leveraged architecture allows to automate the neural architecture search for tiny devices while balancing the accuracy and resource utilization. Despite the huge prospect, existing NAS faces several challenges such as (i) benchmarking, (ii) reproducibility, (iii) transfer learning, and (iv) transferability across NAS-aware tiny devices (transfer whole NAS process or just the final neural model).

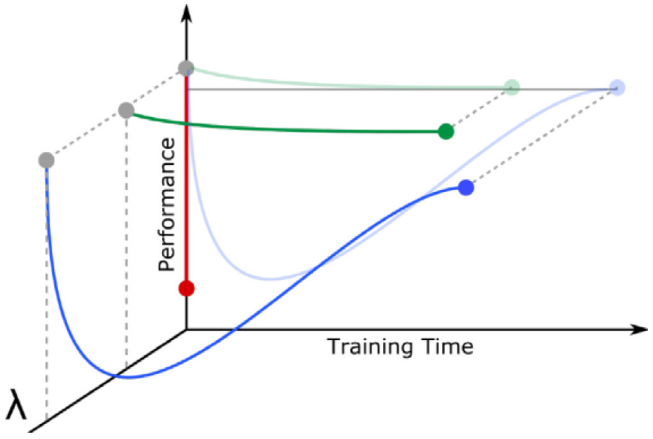

Fig. 13. QGT, depending on λ parameters.

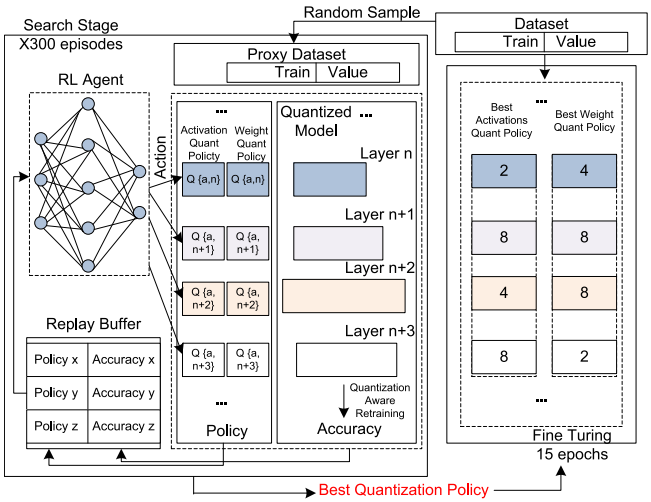

Fig. 14. HAQ framework.

4.6. Model compression

Compression of neural network models can enhance its deployment ability into IoT devices. Various approaches are being investigated to seek for neural model compression for efficient implementation in the IoT ecosystem. In (Signoretti et al., 2021), a tiny anomaly compressor (TAC) model is proposed on top of the typicality and eccentricity data analytics (TEDA) mathematical framework (also known as the empirical data analysis - EDA). The TAC aims to compress user data without prior knowledge about any established mathematical models. Such an approach helps to infer the underlying data distribution for the neural model.

The TAC algorithm runs within a microcontroller where data starts to originate from a sensor. The sensor data is read at some regular interval which is then updated as per the internal metrics. Sample eccentricity is later calculated which is followed by the comparison with eccentricity inequality. A decision is made on this comparison that leads to sample discard (when false) or increment of window count (when true). Window increment value is checked against the full value and discarded when found full. Otherwise, samples are kept and the window is rest for preparation for storage or transmission. Finally, the sample is transmitted to other IoT devices via network connection. TAC algorithm faces major threats from construct validity (verification of relationship between theory and observed value from a study), internal validity (grid-search aware relative linkage to other experiments), and external validity (inferring capability in another context) (see Fig. 12.).

Kronecker Products (KP) has been recently investigated as the key enabler of compression of IoT-based recurrent neural network applications. The study finds 15-38x compression factors when compared to existing methods. However, it is also found that KP provides low accuracy once the same is applied on a large neural network task i.e., language processing by 26%. In (Thakker et al., 2020), a method is devised to recover the accuracy loss by the KP structure for natural language processing. The newly introduced method is called the doped Kronecker product (DKP) compression by using the co-matrix adaptation (CMA) that follows the co-matrix row dropout regularization (CMR) scheme.

Likewise, one can find image compression as a possible application of compression aware machine learning. It is noticed that image compression can lead to poor fit in the IoT ecosystem where network speed is limited. Such lossy nature of image compression happens due to loss of packet in the limited capacity network. Doing so, it reduces the network bandwidth because the lost packet needs to be retransmitted over the network. In (Hu et al., 2020), a design is proposed called *starfish* to achieve better compression ratios in expense of moderate packet loss. In this design, deep neural network architecture is augmented in accordance with an AutoML method for finding suitable TinyML models that can be deployed on the resource constrained IoT devices. In this work, a deep neural network aware streaming framework is implemented for IoT camera application in a low-power wide area network (LPWAN) scenario. The major focus of this design is to provide loss-resilient (interference and collision agnostic) JPEG encoding schemes and direct optimization for the target objective. Currently, a few AI accelerator modules exist in the IoT market such as, AIoT accelerator (K210) and Arduino Uno (ATMEGA 328p). The K210 module consumes 300mW power with 2 MB memory. It is capable of performing image compression at the speed of 230 TOPS INT8. Arduino Uno has 2 KB memory to allow only 8MBPS INT8 processing speed. Both the modules can support the TensorFlow Lite (Micro for Arduino Uno) for image analysis.

4.7. Quantization

Quantization refers to the process of approximating the neural network model that uses floating-point numbers by a low bit width number neural network. The quantization is used to reduce the memory requirement, energy bandwidth, and computational cost for the neural network. However, doing so results in a low precision weight of the neural network. In the domain of TinyML, quantization plays a very crucial role to accommodate the neural network with sacrifice of accuracy. A number of ways are being studied to find a better solution of accuracy loss for the quantization method towards the revolution of TinyML.

In (Chai, 2020), a quantization-guided training (QGT) method is presented for optimization of low-precision targets under the aegis of deep neural network training. QGT is different from standard quantization-ware training (QAT) in terms of utilization of customized regularization for encouraging weight values. It can also identify the compression bottlenecks against 81 KB tiny models for person detection applications down to 2-bit precision. QGT provides a greater advantage than other methods i.e., parameter-based loss functions possess more stable gradients. Despite the fact that quantization-error terms can't be precisely captured, QGT leverages more stable training when compared to generic approaches that rely on the back-propagation gradients. Fig. 13. presents the performance of QGT depending on the λ parameters. QGT can serve either QAT ($\lambda \gg 1$) or post-training quantization (PTQ) ($\lambda = 0$). QGT overcomes the challenges possessed by the standard QAT and PTQ approaches as follows. Significant improvement

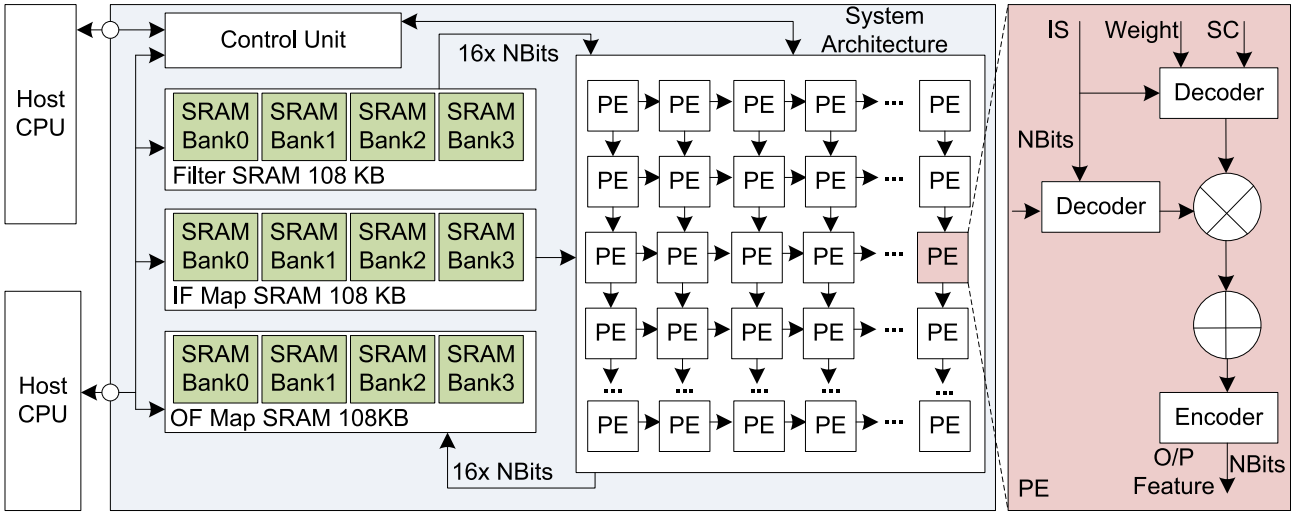

Fig. 15. TENT framework.

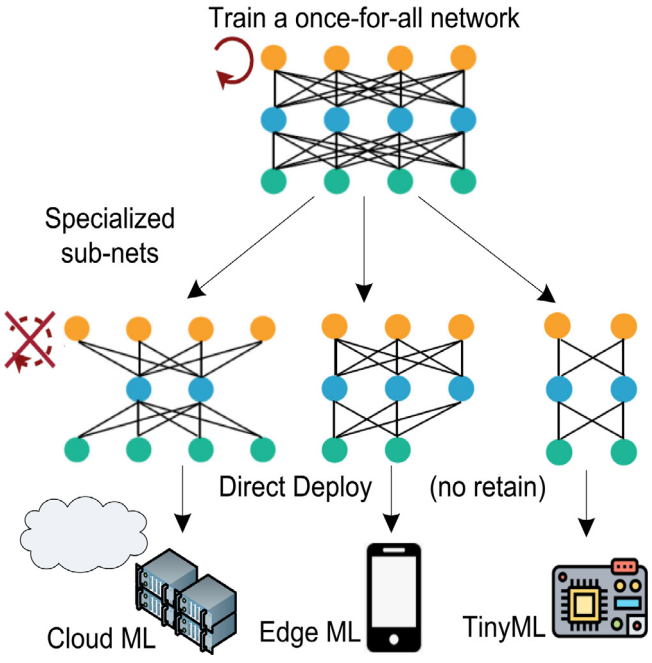

Fig. 16. Once-for-all network training for supporting diverse hardware architectural configurations.

of performance can be observed by QGT even with very little tuning of λ parameters. QGT imposes a soft layer of deep neural network training constraint which could be used to train-aware QAT, PTQ, and weight pruning.

Deep neural network deployment in the tiny MCU has become a challenging issue in recent times. Despite trying an 8-bit quantization scheme the accuracy couldn't be improved at par the expected level. To resolve this issue, (Rusci et al., 2020) presented the HAQ framework which is based on the automated mixed-precision quantization flow scheme. The HAQ framework is targeted to deploy deep neural networks in the tiny IoT-based devices, especially for the reinforcement learning agents (2, 4, 8 bits quantization levels). The HAQ works on the concept of automated precision training to optimally select the quantization bandwidth. While selecting the quantization levels, HAQ aims to optimize the individual weight and activation tensors for a given IoT-based device's memory and processing capability. It works in two-phases, (i) search and (ii) fine-tuning. In the search phase, HAQ uses reinforcement learning agents to follow the given quantization policy. The fine-tuning phase updates the model parameters based on the quantization-aware training process. One quantization policy is selected at a time by a reinforcement learning agent during a number of fixed episodes (300 number). The proxy dataset (subset of main dataset) is fed to the reinforcement learning agent for scoring accuracy and the same is returned back to the reward function. The agent repeatedly (15 epochs) tries to learn the quantization policy and accuracy based on the given proxy dataset. Doing so improves the ability to select the best quantization policy and reduces the accuracy aware vulnerability. The HAQ selects the weight and activation bandwidth during the search phase along with memory constraints associated with the IoT device. Fig. 14. presents the HAQ framework (Wang et al., 2019).

Low-precision integer arithmetic is recently tested in a similar mixed deep neural network aware scenario for efficient deployment in the IoT enabled edge devices (Capotondi et al., 2020). This work presents the CMix-NN to allow flexible and independent tensor quantization for 2, 4, 8 bits weights. The CMix-NN is available at the open -source library aimed for tiny quantized networks.

The library is deployed in the STM32H7 microcontroller with a 224x224 resolution capability and higher accuracy (65% Top1).

We find that a variant of the fixed-point quantization method - TENT is discussed in (Fatemi et al., 2020) that uses the benefits from the tapered numerical format for TinyML models. The TENT method aims to match the numerical format's dynamic range according to the given layer of the deep neural network. This method shows improved results of 31% for energy profiling when compared to ConvNet and ResNet-18 models. In practice, this method emulates the machine learning inference in accordance with quantization. TENT consists of the following, (i) tapered

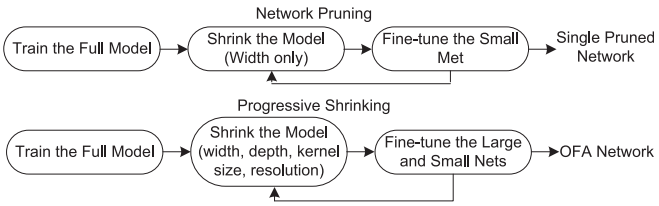

Fig. 17. Progressive shrinking versus network pruning.

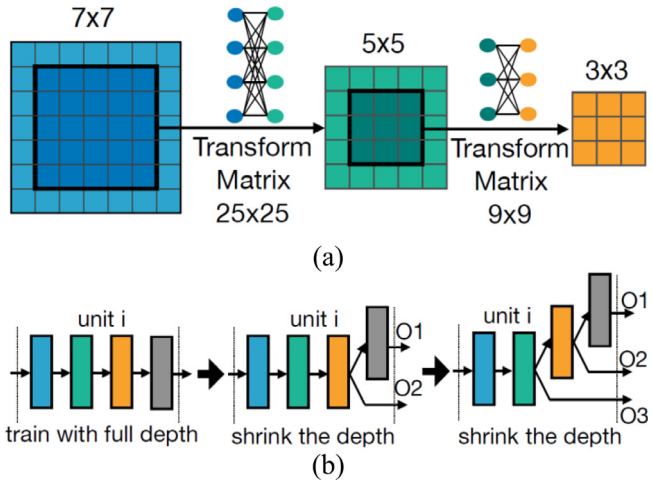

Fig. 18. Progressive shrinking (a) kernel size, (b) depth.

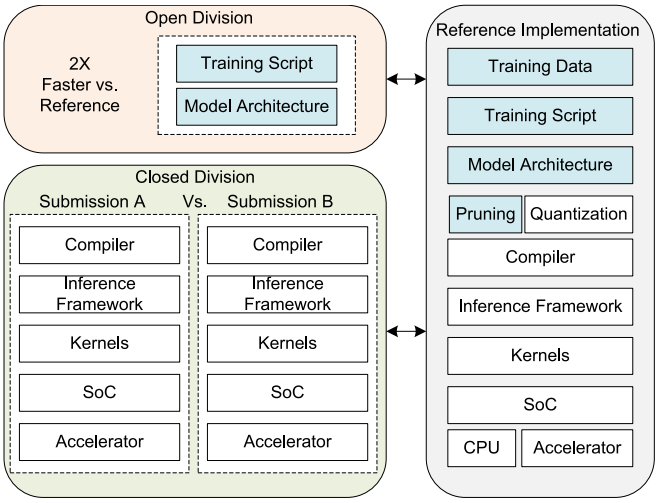

Fig. 19. MLPerf Tiny v0.5 modular design.

fixed-point parameter selection aspect, (ii) low-precision dot product based on the tapered fixed-point, and (iii) quantization paradigm to the fixed-point arithmetic. Fig. 15. presents the TENT framework with custom tapered fixed-point processing elements (PE) for a deep neural network.

4.8. Once-for-all network

Efficient inference remains a challenge for resource constrained edge devices. Existing approaches aim to manually design or apply NAS to find appropriate neural networks for training the network from scratch for *each* case. Doing so incurs huge CO_2 emission in terms of power consumption (e.g., CO_2 emission equivalent to 5 car's lifetimes in USA) (Strubell et al., 2019). A new method is investigated in (Cai et al., 2019) that applies a one-for-all (OFA) network for leveraging a variety of architectural settings where a complete decoupling between training and search is performed. Doing so, overall cost is reduced for each OFA network. The OFA is assisted by the newly developed progressive shrinking algorithm (PSA) that reduces the model size in terms of kernel size, resolution, depth, and width. It is seen that this newly developed PSA outperforms generalized pruning approach for model size minimization. The improved PSA can help to obtain a greater number of sub networks ($>10^9$) which can fit a very large set of various hardware platforms while considering different latency constraints. While doing such model setting, the OSA maintains the same accuracy level which is completely independent of the training phase of the network. The OSA can outperform the accuracy of MobileNetV3 or similar networks with 1.5x fast response time.

The OFA method allows to directly select specialized sub-network for the FA network without requiring training. It can reduce the deployment cost from O(n) to O(1). The method supports flexibility and dynamicity while leveraging various depths, widths, resolutions, and kernels without retaining. The inference is needed to rely on the selective portion of the OFA network for direct deployment for a diverse range of hardware. Fig. 16. shows the OFA deployment. In this approach, the NAS is decoupled from the training stage. Focus is given on the accuracy enhancement of all sub-nets that are derived from a selective portion of the OFA network, during the model training phase. In the model specialization phase, a subset from all subnets is trained to improve the accuracy and latency predictors.

A challenge is noticed during the training of OFA network i.e., a non-trivial task for maintaining accuracy for the huge number of subnets based on the joint optimization of weights. It is computationally infeasible to enumerate all sun-net for getting appropriate gradient during an update step. On other hand, random sampling of sub-nets may incur severe accuracy drops. The main reason behind such behavior is the difference between various subnets that leads to inefficient training of the OFA network. The improved PSA can resolve this issue to retain the OFA network. Under this scheme, the OFA network is not directly optimized from scratch, instead it first trains the largest neural network (having maximum width, kernel size, and depth) and then progressively fine-tunes the OFA network. In this stage, the OFA network is expected to support smaller subnets to share weight with the larger sub-nets. The PSA method provides enhanced initialization while selecting best suitable weights from the larger sub-nets. Distillations of such sub-nets provides an opportunity to minimize the sub-nets to improve the training efficiency. The PSA is a generalized network pruning that aims to shrink depth, kernel size, width, and resolution of the full network while preserving the accuracy of all sub-nets (in contrast, the pruning helps to shrink width of network and accuracy for a single network). Fig. 17. presents the approach behind the network pruning and PSA approach. The evaluation of OFA shows significant improvement of state-of-the-art hardware-NAS while minimizing the CO_2 emission, cost, and GPU hours. Results show that OFA achieves 80% ImageNet top1 with 595 M MACs. Fig. 18. shows the progressive shrinking of (a) kernel size and (b) network depth.

Table 3
MLPerf Tiny v0.5 Inference Benchmarks (Banbury et al., 2021)

Computing Types	Dataset (Input Size)	Model (TFLite Model Size)	Quality Target (Metric)
Keyword Spotting	Speech Commands (49 × 10)	DS-CNN (52.5 KB)	90% (Top-1)
Visual Wake Words	VWW Dataset (96 × 96)	MobileNetV1 (325 KB)	80% (Top-1)
Image Classification	CIFAR10 (32 × 32)	ResNet (96 KB)	85% (Top-1)
Anomaly Detection	ToyADMOS (5*128)	FC-AutoEncoder (270 KB)	0.85% (AUC)

Table 4
MLPerf Tiny v0.5 Round of Submission Summary (Banbury et al., 2021)

Division	Dataset	Training	Model	Numerics	Framework	Hardware	Remarks
Closed	X	X	X	INT-8 PTQ	TFLMicro	ARM MCU	Baseline performance results
Closed	X	X	X	INT-8 PTQ	TFLMicro	RISC-V MCU	Customized neural network inference
Closed	X	X	X	FP-32 & INT-8 PTQ	LEIP	Raspberry Pi 4	Software-only optimization toolchain
Closed	X	X	X	INT-8 PTQ	Syntiant TDK	Neural Network Accelerator	Ultra-low power efficiency
Open	X	QKeras	✓	Int-6/8 QAT	HLS4ML	FPGA	Rapid end-to-end development on reconfigurable fabrics

Table 5
MCUs, Datasets, neural nets for TinyML Benchmark (, xxxx)

	Board Name	Processor, Flash (MB), SRAM, Clock (MHz)
MCUs	Teensy 4.0	Cortex-M7, 2, 1, 600
	STM32 Nucleo H7	Cortex-M7, 2, 1, 480
	Arduino Portenta	Cortex-M7 + M4, 2, 1, 480
	Feather M4 Express	Cortex-M7, 2, 0.192, 120
	Generic ESp32	Xtensa LX6, 4, 0.520, 240
	Arduino Nano 33	Cortex-M4, 1, 0.256, 64
	Raspberry Pi Pico	Cortex-M0+, 16, 0.264, 133
	Name: Feature Dimension, Class Counts	
Datasets	Iris Flowers: 4, 3	Breast Cancer: 30, 2
	Wine: 13, 3	Texture: 40, 11
	Vowel: 13, 11	Drive Diagnosis: 48, 11
	Silhouettes: 18, 4	MNIST Digits: 64, 10
	Aniran Calls: 22, 8	Human Activity: 74, 6
	Name: Topology	
Fully Connected Neural Nets	FC 1 × 10: 1 layer with 10 neurons	
	FC 10 + 50: 1st layer with 10 neurons, 2nd with 50	
	FC 10 × 10: 10 layers with 10 neurons per layer	

After completion of OFA training, the next step aims to derive the specialized sub-net for a given hardware platform. During this phase, a *neural-network-twins* is formed to predict the accuracy and latency for the target network architecture. Doing so, minimizes the need of repeated search cost while substituting the measured cost of already predicted twins.

4.9. TinyML benchmark

Benchmarking allows the machine learning enthusiasts to measure, perform comparison, evaluation and possibly improvement of learning systems. TinyML benchmark is still in nascent stage of development. However recently, the TinyMLPerf (TinyMLPerf, 2021) working group is engaged in facilitation of a set of benchmarking code for the TinyML aware deep neural networks. Such benchmarking allows tiny microcontrollers, neural network accelerators, and digital signal processors (DSPs) to run such benchmarking codes within a constrained speed of 10–250 MHz and consuming as minimum as 50mW of power. The working group has presented the MLPerf Tiny for facilitation of benchmarking for TinyML domain. At the time of writing this paper MLPerfTiny v.5 (MLPerfTiny, 2021) is available that depends on the TensorFlow Lite for Microcontrollers (TFLM) as reference. Though few other benchmarks are still in progress (e.g., EEMBC(r)'s CoreMark(r) and EEMBC's MLMark(r)), they lack in following aspects, (i) non availability of profile for full programs, (ii) lack of accuracy measurement for inference workloads, (iii) needs high memory (>GBs), (iv) more runtime, thus more power.

MLPerf Tiny v0.5 modular design allows comparison of the existing solutions against the reference. Both closed and open division components can be modified with respect to the reference implementation. These two divisions are used to submit results from any organization. Closed divisions leverage direct comparison among the TinyML systems. It also permits for PTQ based on the given calibration datasets; however, no possession of weights is allowed. On the other hand, open division provides an opportunity to improve the TinyML in terms of energy consumption, processing capacity, accuracy, and memory utilization. The submitters in this domain can modify the dataset, model, and the training scripts. Fig. 19. presents the modular design of MLPerf Tiny v0.5.

In Table 3, a comparison between MLPerf Tiny v0.5 inference benchmarks is shown (Banbury et al., 2021). Four use cases are supported by this benchmark that are as follows keyword spotting, visual wake words, image classification, and anomaly detection. Quality target metric is measured % (Top-1) except the ToyADMOS (5*128) dataset which needs area under curve (AUC) metric. Any

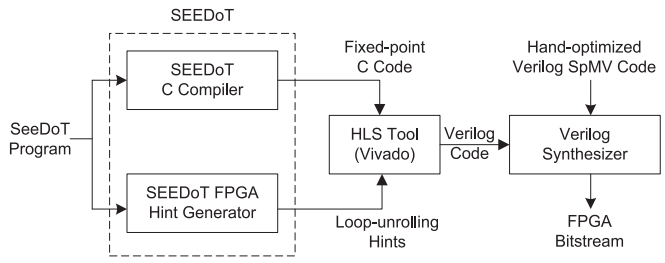

Fig. 20. SEEDoT flowchart.

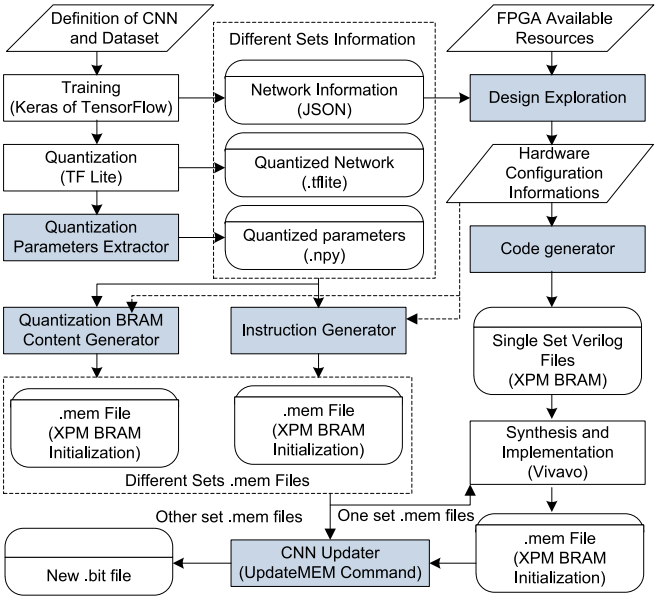

Fig. 21. BRAM-defined toolchain for high-accuracy accelerator for convolution neural network.

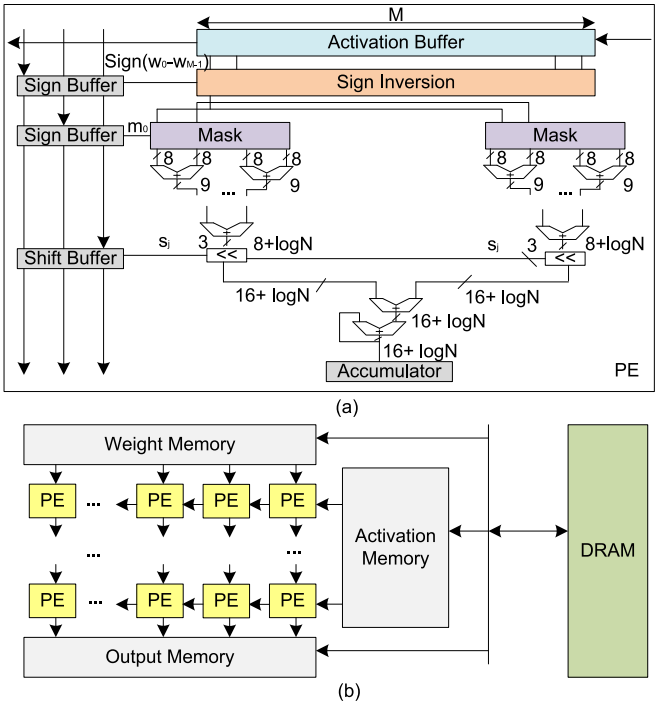

Fig. 22. (a) N-wise bit-serial MAC unit, (b) SWIS workflow.

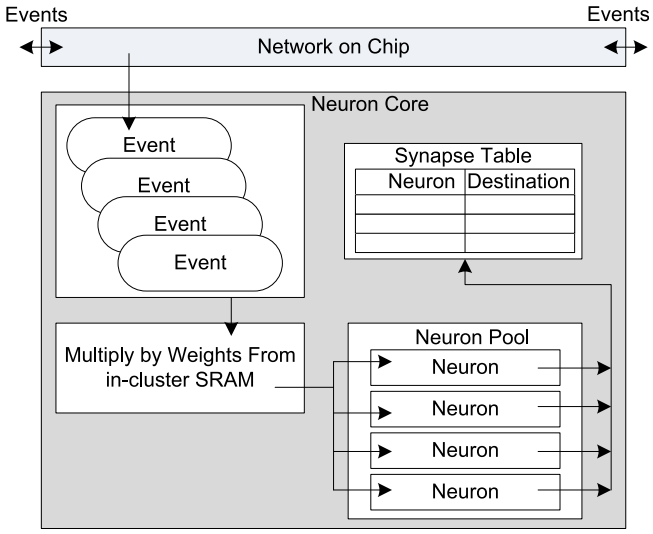

Fig. 23. GrAI One accelerator workflow.

submitter who wants to submit the TinyML model must pass this quality target for each of the dataset. Table 4 presents the summary of MLPerf Tiny v0.5 submissions. Four closed and one open division are mentioned where cross mark (X) is given against some of the attributes (Banbury et al., 2021). The X represents that no modification was made from the reference whereas the tick mark (✓) refers to the modification taken during the PTQ and QAT. It is also noticed that generic versions of numeric format are chosen as 8-bit integers to offer performance boost with minimum impact on the model accuracy. The benchmark allows the TensorFlowLite Micro (TFLMicro) as the key framework under the open-source category and some hardware specific inference compilers. Various hardware platforms are validated in this benchmark that includes ARM and RISC-V. We find the Raspberry Pi 4 is also integrated with the LEIP framework. A workflow is demonstrated by proposing a novel high-level synthesis for ML i.e., hls4ml.

Despite its huge impact in the TinyML domain, the MLPerf Tiny v0.5 has some limitations as follows. It suffers from long-term stability issues at the moment. Addition of new benchmarks should need to be evolved with the time, notwithstanding earlier, can impose a stagnancy in the TinyML design aspects (Banbury et al., 2020). We also notice that the benchmark lacks in the streaming inputs and pre-processing capability.

In (Sudharsan et al., 2021), an alternative benchmark is devised with 3 types of fully connected neural networks, available at (Benchmark-NN, 2021). The benchmark provides 10 datasets with various feature dimensions and class counts. There exist 14 popular hardware boards with various flash, SRAM, and clock speeds where such benchmarks can be tested. The study presents an analysis of 30 neural networks on 7 different embedded boards with minimal power consumption settings for IoT. Table 5 presents the list of all MCUs, datasets, and neural networks available in this benchmark.

4.10. On-Device computing and accelerator

On-device computing and accelerators are gaining popularity in recent times for the field of TinyML to run and deploy neural networks for various applications. In (Gopinath et al., 2019), a domain-specific language - SEEDoT is presented to infer machine learning algorithms for tiny devices. To do that, a compilation technique is developed to reduce search space for any crucial parameter for the fixed-point code. The SEEDoT allows the compiler to compile KB size machine learning models for targeting tiny microcontrollers such as Arduino. The same is extended for floating- and fixed-point FPGA deployments by using a synthesis tool. It can track dimensions of the implied matrices during compile time of the machine learning codes. An associated compiler transforms SEEDoT codes to get converted into fixed-point low-bit width integers. It requires Xilinx's Vivado High-level synthesis (HLS) tool for compiling and hint generation for FPGA. The Vivado helps to use the fixed-point and loop-unrolling hints to get loaded as Verilog code into the Verilog synthesize. An FPGA bitstream is generated by augmented with the existing Verilog code with the hand-optimized Verilog sparse-matrix vector (SpMV) code. Such configuration reduces execution time. The SEEDoT can outperform existing techniques for FPGA and microcontrollers by 3.6–21x and 2.4x–82.2x, respectively. Fig. 20. Presents the SEEDoT flowchart for the earlier mentioned mechanism.

We notice that a convolution neural network accelerator is investigated in (Li et al., 2021d) where a toolchain is demonstrated with high-accuracy block random access memory (BRAM)-aware FPGA oriented flexible structure. The accelerator called – HBDCA integrates the TFL for high-accuracy quantization with 8-bit per-layer of activations. The whole process aims to use the UpdateMEM utility available for Xilinx. Doing so, the content of BRAM can be updated without the need for re-synthesis and re-implementation. The HBDCA can be used to support various kernel-level parallelism such as, 3×3, 5×5, and 7×7 with 2 types of parallelism in fully-connected layers and 4 types at the convolution layers. The HBDCA minimizes the memory access by adopting both spatial and temporal mechanisms for data reuse. The BRAM assisted toolchain is presented in Fig. 21. The Keras TensorFlow Lite is used to train the convolutional neural network and the input dataset. Another input of the BRAM toolchain is the FPGA centric resources used for finding best hardware configuration. Several quantization BRAM aware memory files are generated which are defined as the content of XPM BRAM type. A bitstream file is later deployed and a memory map information (MMI) is generated at the final phase of the flow of the toolchain. The toolchain seems to be suitable for the TinyML environment which avoids re-implementation with high-accuracy work flow.

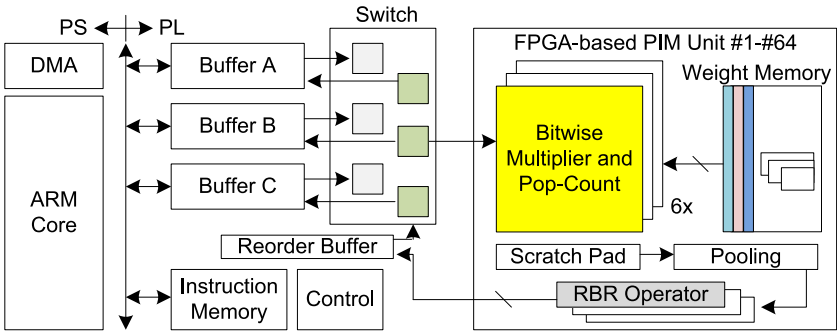

Fig. 24. Processing-in-memory architecture on FPGA.

Shared weight but sparsity aware SWIS framework is presented in (Li et al., 2021c) as a means of quantization for efficient neural network inference acceleration. The SWIS framework improves inference acceleration by applying an offline weight decomposition. A scheduling algorithm is included in the SWIS to achieve 6x speedup and 1.9x energy improvement. The SWIS architecture is aware that the double-shift processing element (PE) is presented as the bit-serial systolic array to provide quantization. This framework allows to integrate minute granularity making the effective number of shifts not limited to even numbers. It presents an improved scheduling algorithm that can harness the fine-tuned tradeoff between the energy and delay of neural network processing. Fig. 22. present the design of PE and the workflow behind SWIS framework.

Several types of sparsity exist in the neural network domain. For example, (i) sparsity in connectivity – neural network has many connection which are not all equal; practically a small fraction of connections improve the computational ability in deep neural networks leaving the space to concentrate only most important connections to avoid large complexity, (ii) sparsity in space – in image and vision processing, a lot of pixels are used to provide right resolution; only a portion of high resolution is used in the images thus giving an opportunity to get rid of majority of the pixels that don't provide valuable information, (iii) sparsity in time – it allows computation only when it is asked to, and (iv) sparsity in activation – a small number of neuron (40%) fire in a deep neural network for propagating the signal from one layer to another, so 60% of energy can be minimized by avoiding useless computation. Thus, sparsity can highly improve the energy consumption and reduce overload during the computation. It seems that sparsity aware designs can significantly be related with the TinyML models. GrAI One accelerator provides such a platform that depends on the sparsity aware computation paradigm. Fig. 23. presents the GrAI One accelerator work flow. The work starts with the events arriving via the network on chip (NOC) centric destination address. The events are then processed in a first-in-first-out queue. The weights are stored in local SRAM for sharing in later phases of flow. The data is weighted and also passed to the neurons. The neurons have states in local SRAM and perform fundamental neural and arithmetic-logic functions. Finally, the events and neural model output values are transmitted to the destinations on the basis of the synapse table.

Accelerator can be used in object detection while featuring processing-in-memory (PIM) framework. Such PIM architectures can be imposed on the FPGAs for energy efficient behavior. Further, memory usage can be drastically reduced under this architectural purview. Similar type of PIM is presented in (Jiao et al., 2021) where a shrunk network is deployed along with a Tiny-YOLO network. The DRAM and look-up table (LUT)-based counters are employed to improve the performance of the underlying DSP hardware. It is shown that the setup can leverage a high throughput of 201.6GOPs at 100 MHz processor frequency. The design eliminates the need of off-chip memory access thus improving processing time with a low-bit-width quantization facility. A total of 64 BRAMs are used along with 3 numbers of RAMs with 876 KB capacity to host the activation buffers. The architecture is developed on a Linux operating system loaded on an ARM core where all memory elements are accessed with the help of AXI bus assembly. An Additional pooling block is included within the space of the RBR block and the scratchpad. Upon arrival of appropriate demand from the next layer, the accelerator can process a reorder buffer for conversion of ordering of data originating from various data flows into a sequential nature (see Fig. 24.).

On-device training and machine learning is an important issue that TinyML research domain must comply with for efficient energy consumption. A number of studies have presented such an approach for improvement of on-device deep learning inferencing with limited memory utilization and minimal processing hardware. For example, in (Disabato and Roveri, 2020), an incremental on-device deep learning is discussed to support the IoT-based

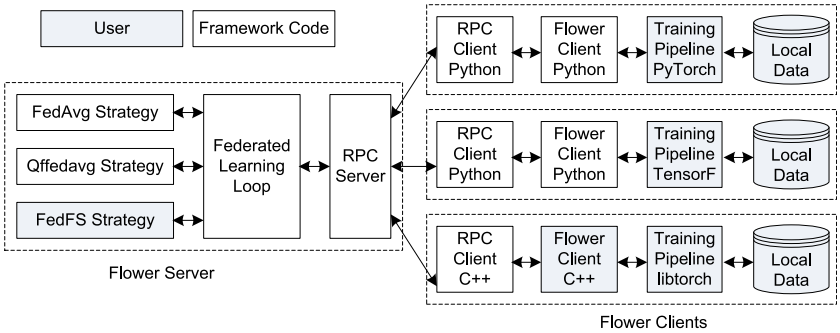

Fig. 25. Flower framework for federated learning at IoT-edge.

ecosystem. For this, the work has used k-nearest neighbor and transfer learning mechanisms. The methods work as follows. Firstly, a data-stream generator generates a sample pair at each instant. Such a sample has unknown probability distribution. The input data along with classification labels is fed to the deep learning algorithm. The whole process relies on the feature extraction and dimensionality reduction mechanisms. The scheme is deployed in Raspberry Pi 3B + and STM32F76I boards for measuring their effectiveness and feasibility.

In another study, we find that model partial replacement strategy is used for data-adaptable tine machine learning scenarios (Kwon and Park, 2021). In this study, a *tiny conv* neural network is deployed where the microcontroller accesses the flash memory for partial update of model candidates. The flash memory contains the interpreter and a dedicated space for storing model arrays where a number of candidate model arrays are considered in the form of partial replacement fashion.

Research shows on-device training for TinyML domain with intervention from the federated learning (Grau et al., 2021). It implements the key word spotting application by using an Arduino Nano 33 BLE Sensor board. Analysis of on-device training is also demonstrated in (Zim, 2021) where ESP32 SoC is chosen for deploying the neural network with a varied number of input of neurons (e.g., 9, 36, 144, 576) with support from the TensorFlow Lite platform and Eloquent TinyML library. It uses the Xtensa LX6 microprocessor platform for conducting this work. In (Estrebou et al., 2021), a similar work is investigated with help of ARM Cortex-M3, Tensilica L106, and Xtensa XL6 where the Scikit-learn enabled algorithms (e.g., XGBoost, support vector machine-SVM, Random Forest, Support Decision Trees, and Gaussian NB) are deployed for comparing accuracy, processing time, and memory consumption.

4.11. In-Processor learning

Federated learning allows to collaboratively train a shared prediction model among the distributed edge devices. In doing so, the data is kept private i.e., at the local edge devices. The federated learning is achieved by repeating following basic phases, (i) update of local parameters on local edge devices, (ii) transmission of localized parameter updates to remote centralized devices for aggregation, and (iii) receiving the aggregated model from the remote centralized device for upcoming local updates. Major disadvantage of federated learning is the minimal intervention of available frameworks that can support such processing on resource constrained IoT-edge devices. Due to severe heterogeneity of computational stacks and unstable network connectivity make the federated learning difficult for resource frugal embedded devices. Several frameworks are being investigated to explore capability of such learning schemes in TinyML aware scenarios. In (Mathur et al., 2021), on-device training is demonstrated under the purview of the federated learning by implying the Nvidia Jetson platform and Android-based smart phone. Fig. 25. presents the Flower architecture where Flower clients can connect with the Flower server by using a remote procedure call (RPC) server. Local data can be trained by using PyTorch, TensorFlow, and libtorch libraries with the help of Flower clients (e.g., Python and C++). The remote Flower server has three strategies namely, FedAvg, Qff Avg (), ad FedFS to enable local edge devices to minimize the burden of training. Thus, minimizing the energy consumption at edges and improving memory capacity. Both *CIFAR-10* and *Office-31* object recognition datasets are evaluated in this framework to compare the model accuracy, energy consumption, and CPU-GPU utilization.

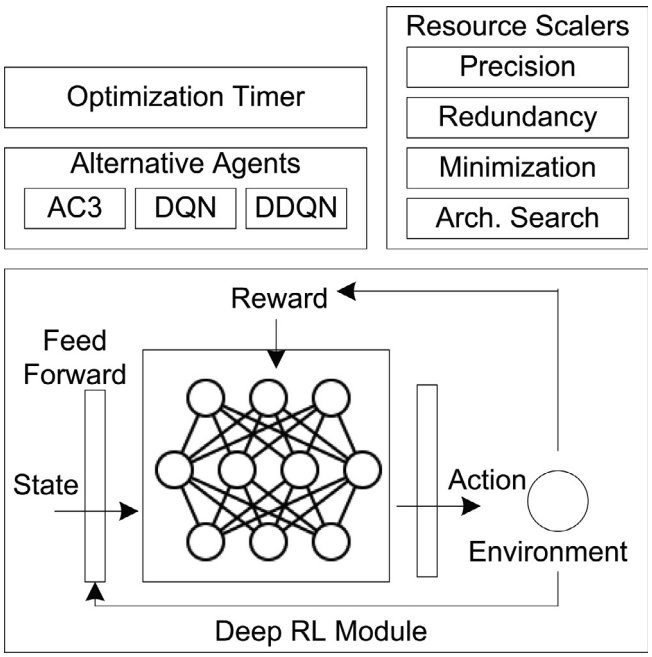

Fig. 26. DQN optimization process.

In (Kopparapu and Lin, 2021), implementation of federated learning is done for IoT-enabled edge devices. The implied *TinyFedTL* approach deploys a machine learning model without needing excess memory utilization. IoT devices running this scheme can re-train various types of classification problems. It uses Arduino to deploy the *TinyFedTL* on top of Tensorflow Lite Micro with an on-device training facility. Both fully connected and backpropagation updates are made possible coded in C++. Such implementation provides security, privacy and responsible AI-aware design spectrum for TinyML domain.

We notice another work on on-line (on-device) deployment of machine learning algorithms by using Arduino board in (Ren et al., 20211a). It deploys the *TinyOL* method to treat the learning model as a static object. Retraining of streaming data is supported by an additional layer called *update layer* in this approach. The TinyML model can be fed as scaled input to the update layer which updates the model as per the metrics and simultaneously makes a prediction. The TinyOL method runs on the microcontroller, however the TinyML model is fed into the MCU externally. Two use cases, (i) fine-tune and (ii) multi-anomaly classification is performed under this setup to validate this efficiency of the mentioned scheme.

Studies have performed other options besides federated learning to seek for efficient implementation of deep learning models in IoT-edge devices. For instance, in (Li et al., 2021a), an architecture called *Rosler* is presented to showcase a compressing-while-training framework. The scheme helps devices to learn with less memory utilization. It attains 50 $\sim$ 90x (16-bit quantization) when tested against various datasets. The major backbone of this technique lies in the approximation of rank-restricted back-propagation algorithm so that compression ratio can be extended to enhance the butterfly-folding (BUFF) method. By doing so, miniscule weight matrix is learned which requires less memory.

Resource optimization for deep reinforcement learning has become an interesting concept where decisions are made on the basis of the inputs. In (Svoboda et al., 2020), a Deep Q-net (DQN) reinforcement model is leveraged to minimize the memory constraints in the IoT-edge devices making it suitable for TinyML applications. Fig. 26. presents the DQN optimization process where deep reinforcement learning modules and resource scalers work together. The optimization tuner finds how to combine the individual scalers (i) redundancy minimization, (ii) mixed-parameter pre-

Table 6
Comparison between TinyML Frameworks by Various INCs (Sanchez-Iborra and Skarmeta, 2020)

Framework	Compatible Hardware	Interoperable External Libraries	Output Language	Algorithm	Open Available
TensorFlow Lite	ARM Cortex-M	TensorFlow	C++ 11	Neural Network	✓
Weka-porter	Not Fixed	Weka	C, Java, JavaScript	Decision Trees	✓
ELL	ARM Cortex-A, Arduino, Micro:bit	ONNX, CNTK, Darknet	C, C++	Neural Network	✓
TinyMLgen	ESP32, ARM Cortex-M	TensorFlow Lite	C	Neural Network	✓
uTensor	mBed	TensorFlow	C++ 11	Neural Network	✓
ARM-NN	ARM Cortex-A, ARM Ethos, ARM Mali	ONNX, TensorFlow, Caffe	C	Neural Network	✓
STM 32Cube AI	ARM Cortex-M	PyTorch, Caffe, TensorFlow, ConvNetJs, Lasagne	C99	Neural Network	✓
MicroMLGen	ESP32, ESP8288, Arduino	Scikit-learn	C	RVM, SVM	✓
CMSIS-NN	ARM Cortex-M	PyTorch, TensorFlow, Caffe	C99	Neural Networks	✓
emlearn	ESP8266, AVR ATMega	Scikit-learn, Keras	C	Bayes, Neural Networks, Naïve Gaussian, Decision Tress, Random Forest	✓
m2cgen	Not Fixed	Scikit-learn	PHP, Go, C, C#, Java, R, Dart, Python, JavaScript, Visual Basic, PowerShell	Random Forest, Neural Networks, Linear Regression, SVM, Logistic Regression. LGBM Classifier, Decision Tress	✓
AlfES	ARM Cortex-M4, Arduino, Raspberry Pi, STM32, Windows (DLL), STM32 F4, ATMega32U4	Keras, TensorFlow	C	Neural Networks	X
NanoEdge AI Studio	ARM Cortex-M	–	C	All Unsupervised learning	X

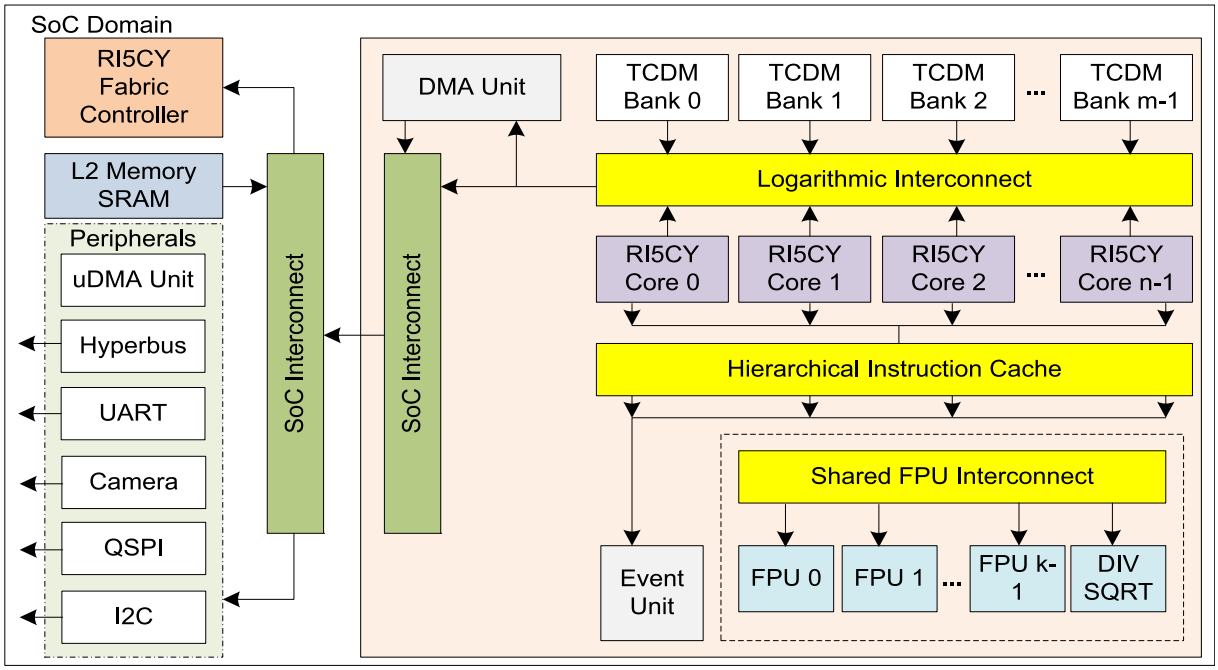

Fig. 27. PULP framework.

cision, (iii) convolution optimization, and (iv) architecture search to make the tuning tractable.

A study shows that peak memory explosion can be minimized in embedded systems by using the Raster-Scanning-Network (RaScaNet) for image processing applications for the TinyML ecosystem (Yoo et al., 2021). This network reads a few rows in the pixel matrix. Later, it learns the representation of the full image by using the recurrent neural network. It requires no reduction of input image size. The study shows that the network needs 15.9–24.3x lesser peak memory. The weight memory is also less required i.e., 5.3–12.9x than existing tiny models. The on-chip SRAM is captured by this network that needs a maximum of 60 KB of capacity.

5. TinyML frameworks

5.1. TinyML frameworks by INCs

TinyML frameworks are being developed by some incorporations (INCs), particular developers, and research groups. Table 6 presents the comparison between some identified frameworks developed by INCs and particular developers (Sanchez-Iborra and Skarmeta, 2020). Majority of such frameworks are publicly available except AlfES and NanoEdge AI Studio which are developed by Fraunhofer IMS and Cartesiam. We notice that most of the frameworks support ARM Cortex family. Some support ESP8266,

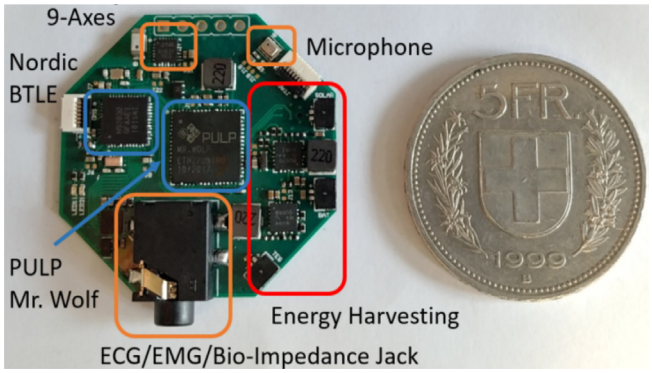

Fig. 28. InfiniWolf prototype for FANN-on-MCU experiment.

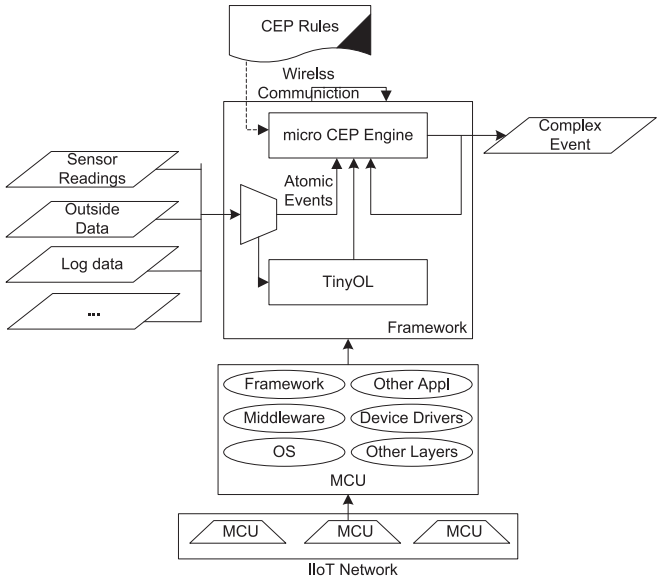

Fig. 29. System framework for complex event processing in the industrial IoT environment.

ESP32 family hardware. A few supports Arduino and Raspberry Pi. STM32 is supported by AIFES. There is no such fixed requirement for m2cgen and Weka-porter frameworks. The Micro:bit is supported by the ELL framework which originated from Microsoft. The uTensor allows mBed to run the neural network. We also notice that C and C++ are mostly supported output languages from such frameworks. Several external interoperable libraries are in use with the frameworks. For example, TensorFlow, TensorFlow Lie, Keras, Scikit-learn, PyTorch, Caffe, ONNX, Darknet, CNTK, Lasagne, ConvNetJs, and Weka are used as interoperable external libraries.

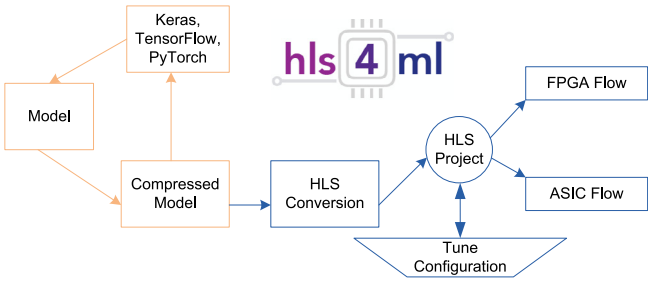

Fig. 30. hls4ml framework.

5.2. TinyML frameworks by research groups

Several frameworks are presented by various research groups around the world to implement TinyML models in resource frugal devices. Deploying TinyML models in edge devices can minimize latency and improve privacy. In (Tabanelli et al., 2021), a parallel ultra-low power (PULP) framework is discussed for IoT-based processors. The PULP allows to run non-neural machine learning kernels that can compete with neural networks in terms of accuracy. The PULP framework is tested against the PULP-OPEN hardware which shows 12.87x faster than ARM Cortex-M4 MCU. The PULP is capable of running SVM, linear regression. K-means algorithm, random forest, k-nearest neighbor, and Gaussian naïve bayes algorithms. Fig. 27. presents the PULP platform running on the system-on-chip (SoC). It can feature the RI5CY core in the SoC domain along with L2-SRAM. The fabric controller controls several cores of tightly coupled data memory (TCDM). Doing so, the PULP minimizes overall energy consumption. Cluster domain of PULP contains a logarithmic interconnect and shared floating-point unit (FPU) interconnect to enable TCDM to communicate with the RI5CY cores and FPUs.

An open-source toolkit i.e., FANN-on-MCU is developed to minimize energy consumption (Wang et al., 2020). The framework runs on the PULP platform. The speedup is increased by 22x and 69% energy is less consumed when compared to the RISC-V octa-core processor. The FAN-on-MCU is built upon the fast artificial neural network (FANN) library for running the framework on light-weight IoT-aware devices. The framework is implemented in the InfiniWolf prototype as shown in Fig. 28. Such a prototype can be kept always-on for running machine learning algorithms. The prototype is coupled with multi-channel I/O direct memory access (µDMA) along with 512 KB L2 cache in the SoC domain. The cluster domain has 4 KB SRAM L1 multi-bank memory for parallel access with 8 RISC-V cores. The InfiniWolf runs on a wearable battery for accessing multiple sensor data (e.g., 9-axis motion sensor, pressure sensor, microphone, and ECG/EMG analog front end). A smart power supply unit (PSU) uses the BQ25505 tribo-electric generator (TEG) aware energy harvesting technology. A BQ25570 solar energy harvester along with a fuel gauge (BQ27441) with 1.8 V LDO. It runs on a 120mAH Li-Ion battery. The data communication is wirelessly controlled by nrf52832 Bluetooth. The process flow of this framework follows four basic steps, (i) obtaining dataset, (ii) preprocessing (normalization of the data and conversion to FANN format), (iii) define and train neural network (explore hyperparameters and find best network), and (iv) deployment on the embedded device (convert neural network to fixed-point, integration with sensors read-out, and measure power consumption/performance).

Complex event processing (CEP) is a trivial task for industrial IoT (IIoT) scenarios. It gets more cumbersome when machine learning needs to be associated with it for edge aware decision making. In (Ren et al., 2021c), a framework is presented that can exploit the synergy between machine learning and the CEP at the IIoT ecosystem. In such a framework the computation is shifted from cloud to the local edge devices allowing users to adapt the on-device model inference. The IIoT needs a number of key attributes (e.g., local device intelligence, distributed computation, energy consumption, and flexibility) for provisioning efficient services. Fig. 29. presents the CEP framework that is supported by TinyOL. The micro CEP engine is governed by the CEP rules along with atomic events that originate from the sensor readings, log data and outside data. The micro CEP engine is asserted with the MCU-aware involvement from the IIOT network.

Minimization of energy consumption has become an important factor for deep neural networks under the TinyML ecosystem. In (Dogaru and Dogaru, 2020), a light binary convolutional neural

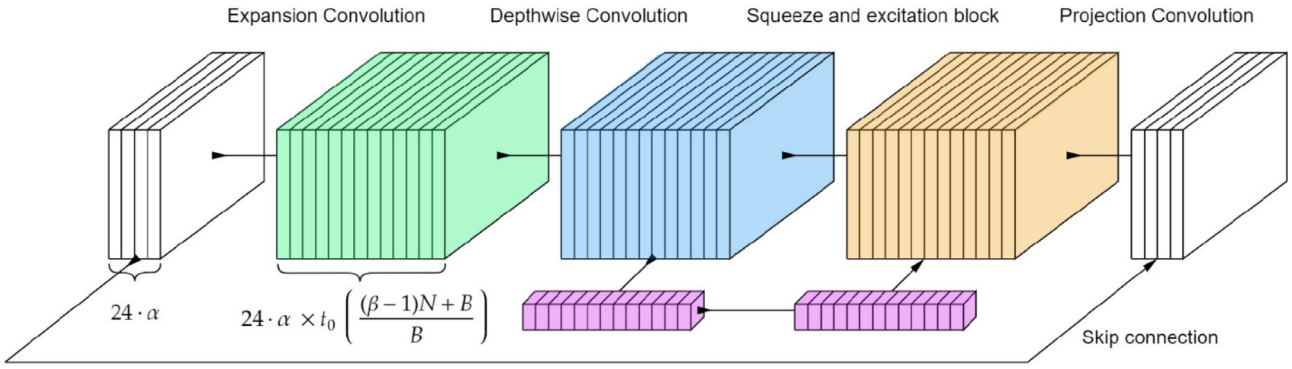

Fig. 31. *PhiNets* convolution block framework.

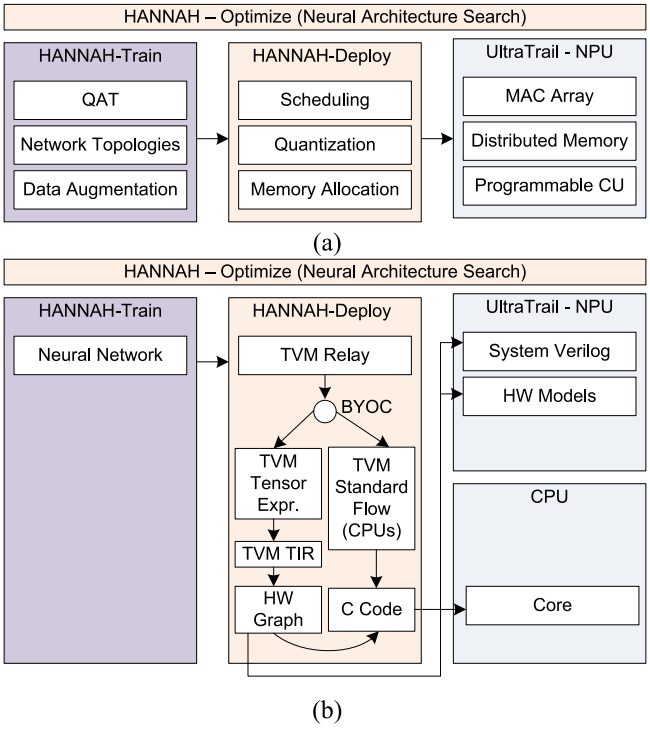

Fig. 32. (a) HANNAH framework, (b) TVM-based HANNAH workflow.

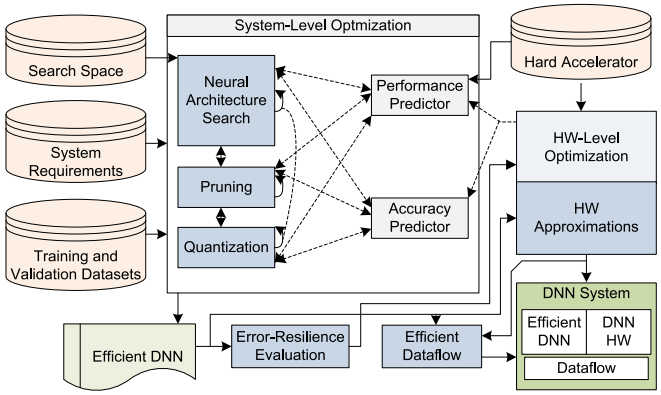

Fig. 33. Cross-layer EdgeAI framework.

network (LB-CNN) framework is presented to achieve this goal for industrial applications. LB-CNN is open to developers that can be accessed in standard .h5 format. It uses the Chainer/Cupy library to offer training to the output layer. Keras or TensorFlow are also available at the output layer to improve the accuracy. It is supported by an extreme machine learning (ELM) approach for fixed-point quantization. The framework deploys depth-wise convolution max-pooling, and dense layers on top of binary kernels.

In (Fahim et al., 2021), a co-design workflow aware open source TinyML framework – *hls4ml* is presented for minimization of energy consumption. The hls4ml enables scientists to accelerate machine learning aware FPGA and ASIC implementation feasible and easy. It can provide pruning and quantization-aware training for low-power embedded devices. Python-based APIs are made available for harnessing the scientific benefits out of the framework. Fig. 30. presents the hls4ml framework where red boxes refer to machine learning model, compression, and optimization. The blue portion involves high-level synthesis or HLS conversion, configuration tuning and FPGA/ASIC flow. It has received support from the various INCs (e.g., Xilinix, Intel and Mentor) for leveraging end-to-end machine learning implementation.

In (Paissan et al., 2021), a scalable backbone framework for deep neural networks is presented – *PhiNets*. The PhiNets framework is designed to provide image processing application support for resource constrained IoT-edge devices. It is built on top of inverted residual blocks for decoupling memory, cost, and over-processing. The framework is associated with the YoloV2 detection head. It can minimize parameter count by 87–93% when compared to EfficientNetv1 and MobileNetv2. The framework is deployed on STM32H743 MCU having only 2 MB flash and 1 MB SRAM. It can process the neural network by consumption of 10mW power. Fig. 31. presents the framework behind PhiNets. It has 24α number of filters in the first bottleneck layer. The α refers to the hyperparameter with resemblance to MobilenetV2 architecture. The framework has the property to down sample the feature map when the multiplication factor is doubled. The Squeeze-and-Excitation (SE) blocks are inserted after every convolution layer. The expansion factor t is represented by $t_0(\frac{(\beta-1)N+B}{B})$, where β refers to shape hyperparameter, t_0 is the base expansion factor, N is the index of each inverted residual block starting with 0 for the first layer. The number of channels is initially increased with pointwise convolution. Then, a depthwise convolution is done followed by the SE block. Lastly, a second pointwise convolution is connected to the low-dimensionality bottleneck block. It can mitigate the hardware aware scaling by optimizing multiply-accumulate operations (MACC) and working as well as parameter memory of IoT devices.

Hardware/software co-design is another issue that should be resolved in order to harness the TinyML aware service for edge-AI enablement. In (Bringmann et al., 2021), such a hardware accelerator and neural network search (HANNAH) framework is presented. The aims of HANNAH are to automate the

co-optimization process of a neural network architecture for efficient end-to-end deep neural training and deployment at the edge devices. The framework follows three steps, (i) training, (ii) deployment, and (iii) association with the UltraTrail neural processing unit. The training phase consists of quantization aware training (QAT), data augmentation, and topology information. The deployment phase invokes scheduling, memory allocation, and quantization aspects. Finally, the UltraTrail – NPU association incurs the array of the multiply-and-accumulate (MAC) units which is configured for the TC-ResNet. The TVM-based HANNAH work flow is shown in Fig. 32. Firstly, a neural network is trained which is then passed to the deployment phase where it is transformed into the RelayIR. A graph partition is then performed into UltraTrail – NPU supported hardware operations. Required memory layout transformation is carried out in this phase of workflow. A hardware graph is built from the lowered TensorIR (TIR) that forms the basis of the deployment. The same is used to model and synthesize the hardware. A C-Code emission is made via the bring-your-own-codegen (BYOC) framework. Later, the register-transfer level (RTL)-Code is generated for actual execution in the TVM-aware target system.

A cross-layer optimization framework is presented in (Shafique et al., 2021) to consider the accuracy and performance for exploring architectural space. The aim of the framework is to generate a suitable Pareto-optimal deep neural network model. The generated model is later optimized based on quantization and pruning techniques. The optimized neural model is next considered for error-resilience analysis. Doing so, it helps to guide the hardware (HW) approximation stage to minimize energy consumption. An efficient data flow strategy is lastly invoked along with the layer partitioning scheme to optimize the off-chip i.e., external memory access, A scheduling of memory access is simultaneously devised for reducing the memory utilization for weight activation storage. Fig. 33. presents the EdgeAI framework for optimizing the cross-layer behavior. The framework leverages the software, hardware, and dataflow level optimization.

5.3. Use cases in TinyMl

Several use cases can be demonstrated by using TinyML. The list of use cases can include following, but not limited to image recognition, speech recognition, sign language processing, phenomics, face detection, hand gesture recognition, few shot keyword spotting, precision agriculture, sea turtle conservation, cough detection, body pose estimation, ecology monitoring, traffic control, environmental prediction, respiratory symptoms, autonomous vehicle monitoring, and always-on-voice wakeup.

5.4. Speech recognition

Speech recognition is considered as a generic application for any machine learning model. TinySpeech is such a framework that can provide low-precision on-device speech recognition (Wong et al., 2020). TinySpeech uses attention condensers for leveraging low footprint memory utilization and minimal processor power consumption. This framework is tested against the Google Speech Commands benchmark for validation of its working. It is found that it significantly minimizes the architectural complexity by 507x in terms of parameters. It can provide 48x less multiply–add operations. Memory requirement is very less i.e., 2028x lesser when compared with earlier published works. It can detect limited-vocabulary speech recognition. The framework takes the mel-frequency cepstrum coefficient (MFCC) representation from an audio signal as an input. It comprises a convolution layer, an average pooling layer, a fully connected layer, and a softmax layer. It shows high diversity in form of architectural point of view. It uses attention condensers and sparsely use generic stand-alone convolution modules also no batch normalization is carried out.

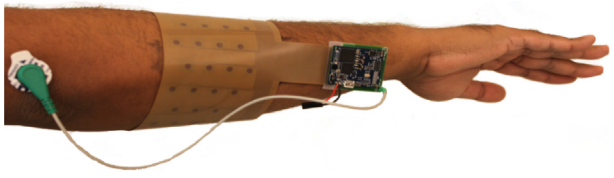

Fig. 34. Sensor fused hand signal acquisition device with flexible substrate design.

5.5. Image recognition

Deep image recognition is performed in (Wong et al., 2020) by using visual attention condensers. The implied AttendNets is presented as a low-precision on-device image recognition framework. It uses machine-designed micro–macro architectures which are harnessed from a machine-driven design process. It is tested against the $ImagNet_{50}$ benchmark for showcasing the significance of the AttendNets. The results show that it lowers 3x multiply–add operations and improved 7.2% accuracy. It also uses 4.17x less parameters and 16.7x minimized weight memory when compared to MobileNet-V1. The framework presents an optimal choice between the gap of accuracy and network efficiency. It improves the spatial-channel selective attention by using automated micro–macro architectural aspects. The usefulness of such incorporation can enable on-device image recognition as a key enabler of TinyML application for low-cost and low-energy aware IoT systems.

5.6. Sign language prediction

TinyML can accommodate the sign language prediction use case. In (Paul et al., 2020), an American sign language prediction aware architecture is presented. The architecture can work on an ARM Cortex-M7 MCU having 496 KB of memory footprint. The implied model size is only 185 Kb during post-quantization phase and can infer 20 frames/s. The work uses four datasets namely, Sign MNIST, Kaggle ASL, OpenMV H7 aware dataset, and ASL Alphabet Test dataset all from Kaggle. A generalization is aimed to predict for ASL fingerspelling with 98.84% test accuracy. It is also suggested that further accuracy drop can be managed by using interpolation where output is expected to be float32. The generalizability of 74.58% is achieved in this study and can be exploited at any ARM MCUs. Despite the fine results, it seems that the generalization accuracy should be improved by using dataset from

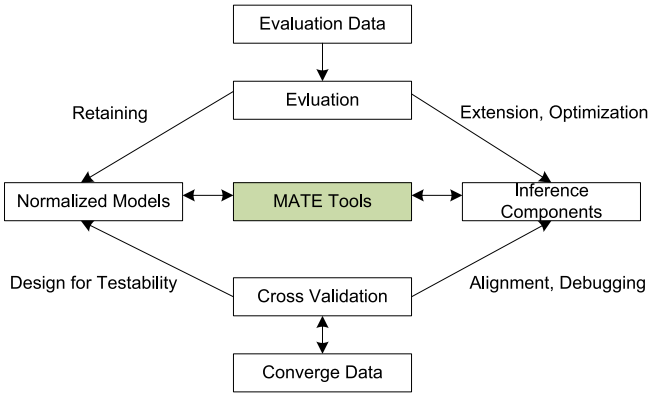

Fig. 35. MATE framework.

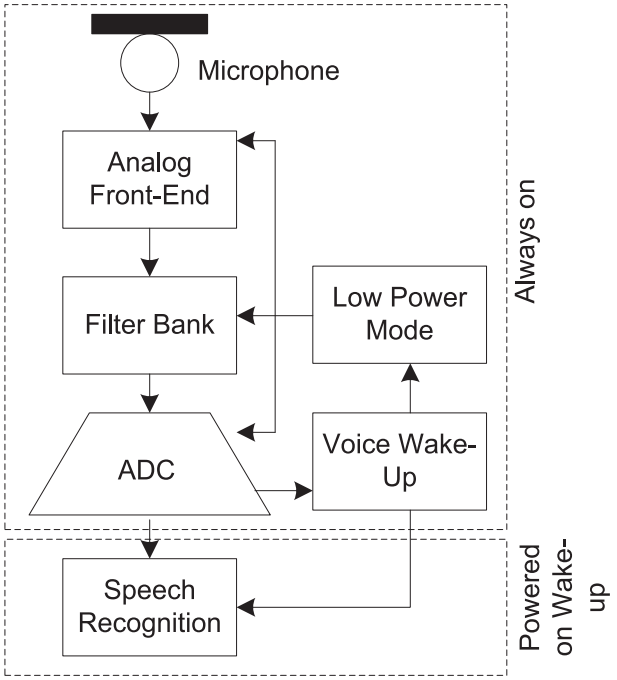

Fig. 36. Always-on-voice wakeup system model.

diverse sources. More complicated sign languages should be further investigated for more generalizability enhancement.

5.7. Hand gesture recognition

Hand gesture recognition is a promising field of the TinyML domain. Several methods are investigated to apply hand gesture prediction in resource constrained devices. For example, hyperdimensional computing is considered for implementation of electromyogram pattern recognition (Zhou et al., 2021). In this study, limb position aware gesture prediction facility is investigated by using sensor fusion technology, The accelerometer and EMG signals are fused together for emulation of dual-stage classification. Position specific parameters are kept in superposition to multiple models. It gained 93.34% accuracy while validating the model when checked against a dataset with 8 limbs and 12 gestures. Results show 8x less memory usage for this work. Fig. 34. presents the sensor fusion of EMG and accelerometer sensors for capturing hand signals.

Capacitive sensing has been considered to predict hand gestures for TinyML settings (Bian and Lukowicz, 2021). This study devises a wrist-worn embedded prototype having 4 capacitive sensing electrodes. It uses only a hidden layer to act as the classifier with an accuracy of 96.4%. The design uses an ARM Cortex-M4 MCU with 1 MB flash and 256 KB RAM. Finally, the deployed on-board model has only 29.6 KB size in float32 mode without quantization. The inference time is very less i.e., 12 ms with 26.4mW power consumption. Another work demonstrates an NRF52-like processor with a 32 Hz accelerometer attached on the first finger. It chooses a multi-layer long-short term memory (LSTM) model into the TensorFlow Lite Micro libraries with 2.8 MB on-disk memory size. The accuracy is stable and measures 95–95% per gesture out of 10 various gestures.

5.8. Body pose estimation

Body pose estimation can play a vital role for monitoring elderly health care. In (Vuletic et al., 2021), a platform agnostic framework

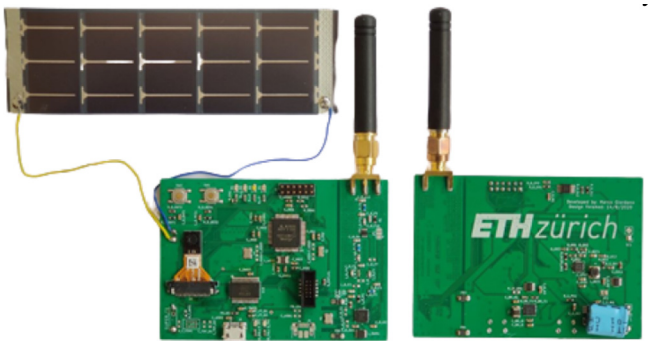

Fig. 37. Sustainable IoT-based face detection sensor system.

is presented that enables validation and fostering of model-to-platform adaptations rapidly. It uses following algorithms face-landmarks (e.g., RetinaFace) and body pose estimation (e.g., OpenPifPaf – uses Composite Fields for real-time detection of spatio-temporal body pose) for deploying the application in the Nvidia Jetson NX platform (consists of GPUs, deep learning hardware accelerators). The implied model audit toward endorsement (MATE) framework (see Fig. 35.) can provide extensible inference to the machine learning model. It also supports image transformation in such a way as to deploy the same model to other platforms too. The use case is named as *Kestrel* to monitor the users' body pose during home or hospital. If the user falls off their safe position, an alarm is raised. The TensorRT library is integrated with the MATE for easing the unification of inference-engine across various access layers.

5.9. Few-Shot keyword spotting

TinyML can help to easily train and deploy the keyword spotting (KWS) for use by numerous speakers and a huge variety of accents for each word. A few-shot KWS is deployed in (Mazumder et al., 2021) that uses only 5 training examples which could be used for any language. The procedure obtains an F1 score of 0.75 for 180 new keywords for 9 different languages. Such an embedding model can achieve a reasonable F_1 score of 0.65 on just 5 shots. The 5-shot model is tested for accuracy in following two areas, (i) keyword search and (ii) keyword spotting. It is found that keyword spotting accuracy is 87.4% for 22 languages with 440 keywords. The 5-shot KWS model uses a 49x40 spectrogram and embedding representation. It uses a softmax layer to classify 3-categories with 5 targets and 128 non-target samples (unknown).

5.10. Always-on-voice wake up

Voice activity detection (VAD) is becoming a popular application in the IoT-based domain. TinyML can support such use cases by using a speech start approach. For example, in (Hardy and Badets, 2021), a recurrent neural network aware classifier is developed to enable always-one voice detection by implying a wake-Up sensor (WUS). The approach shows less than 3% no trigger rate (NTR) during less than 1% dirty cycle. The power consumption of WUS is just 45nW for the recurrent neural network with 530 Bytes of memory footprint. Fig. 36. presents the always-on WUS system model. The underlying classifier analyzes the digitized short-term spectrum as fed input of audio signal at a given rate. Such spectrogram analysis detects the state of the speech i.e., starting. Upon detection of starting speech, the WUS initiates the speech recognition engine with the help of a low noise amplifier (LNA). A set of band pass filters (BPF) extracts the audio signal spectrum with central frequency 0.1-7KHz range. Difficulty is perceived during detec-

tion of speech that starts with 'f', 's' and 'sh'. The work uses a noise section from the MUSAN dataset and speech segments from the Speech commands dataset. The work uses quantization of minimal gated unit (MGU) with recurrent neural networks.

5.11. Face detection

Face detection application is an important use case for the IoT-based smart ecosystem. Battery-free long-range pervasive face detection can certainly improve human face monitoring in various applications. In (Giordano et al., 2020), an ARM Cortex-M4F MCU is integrated with a low-power camera module (e.g., HM01B0) and a time-of-flight sensor (e.g., VL53L1X). The system communicates to a remote system by using the LoRa transceiver module (e.g., SX1262). The system power is managed by the BQ25504 chip. Initially, the time-of-flight module is powered-on on an automatic basis (e.g., 1 Hz). If it detects any movement, the MCU is triggered to wake-up. The camera produces images of 320x320 pixels. Both VGA and QVGA modes are possible in this setup. That consumes as low as 1.1mW of power at 30 frames/s. Two convolution layers are used with two convolution-MaxPool layers in this setting. Three dense layers finally help to detect faces. The work uses CelebA dataset for training the face classifier based on the TensorFlow framework on the MCU. It achieves 97% accuracy in the wireless camera sensing design mode. Fig. 37. presents the IoT-based sustainable face detection system. Face mask detection has become essential in the emergence of the COVID-19. Several research groups have attempted to develop TinyML aware face mask detection. For example, (Mohan et al., 2021) deploys a convolution neural network on ARM Cortex-M7 MCU clocked at 480 MHz to perform face detection by using 496 KB buffer memory. The model is trained by using the TensorFlow framework that incur 138 KB memory during post-quantization at 30 frames/s. In (Luukkonen et al., 2021), a cough activated face visor is presented. The face visor can be opened and closed when a cough sound is detected.

5.12. Cough related respiratory symptoms detection

COVID-19 has suddenly put the world in a cautious mode where cough related respiratory symptom detection has become an important task. In (Rashid et al., 2020), a convolution neural network aware model – *Tiny RespNet* is presented. The model is scalable and works on multimodal settings. The setting is deployed on the Xilinx Artix-7 100 t FPGA with a facility for parallel processing. The power consumption is low i.e., 245mW and energy efficiency is

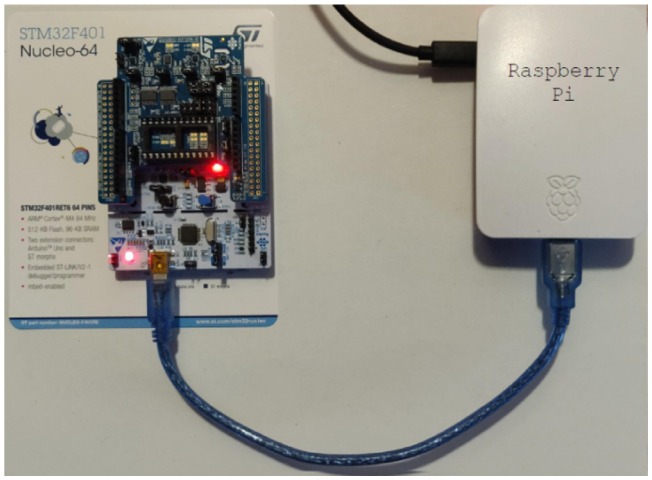

Fig. 38. DTNN deployment for NWP.

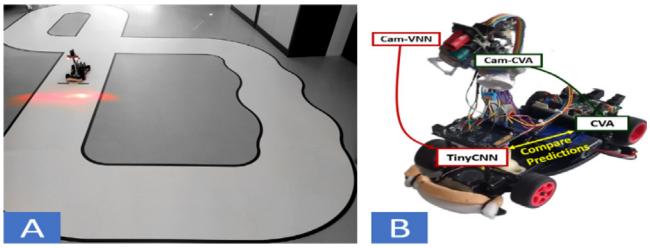

Fig. 39. Mini vehicle driving, (a) running on closed-loop track, (b) closed-loop learning task.

4.3x better than existing alternatives. The model is also implemented in the NVIDIA Jetson TX2 SoC and compared against the processing capability of TX2 CPU. The Tiny RespNet framework can take input from audio recording, patient speech, and demographic information and perform classification. Cough detection related respiratory symptoms are classified based on the three datasets e.g., ESC-50, FSDKaggle2018, and CoughVid. The framework can detect the dyspnea. Another work shows the cough detection application by using Arduino Nano BLE Sense platform with support from the EdgeImpulse cloud (Cough detection, 2021).

5.13. Phenomics and ecological conservation

Phenomics refers to the study of phenotypes of genome-wide variations in the lifespan of an organism. Especially, the plant phenomics uses the genetic improvements for discrimination of important germplasm that possess valuable traits among the whole set of germplasm. In (Nakhle and Harfouche, 2021), a phenomics image analysis is performed based on the classification of tomato leaf disease and spider mites. It uses the PlantVillage tomato dataset along with YOLO3 (based on DarkNet-53 architecture) algorithm for automatic detection of tomato leaves. The study also uses SegNet algorithm for pixel-wise image segmentation. It further investigates several other commonly known data analysis tools to validate the use of phenomics under the aegis of TinyML paradigm.

Ecological conservation analytics has seen a growth by using AI aware techniques in recent times. In (Curnick et al., 2021), an ecology conservation task is deployed by using TinyML in the SmallSats (e.g., small payload satellites) scenario. Sea turtles getting abolished due to unplanned fishing and sea pollution. Conservation of sea turtles is recently associated with the TinyML framework by using the computer vision domain. The study aims to improve the conservation of sea turtles by using state-of-the-art real-time vision-based TinyML supports.

Environment monitoring is another crucial application that can be integrated with the TinyML architectures. In (Alongi et al., 2020), a deep tiny neural network (DTNN) is presented for numerical weather prediction (NWP). The framework uses STM32 MCU and X-CUBE-AI toolchain on top of the Miosix operating system. It requires only 45.5 KB of flash and 480 Bytes on-board RAM to run the DTNN architecture. Fig. 38. presents the system deployment for the weather prediction by using a Raspberry Pi.

5.14. Autonomous vehicle

Autonomous mini vehicles are used in various emergency scenarios such as military, fire-fighting, human tracking under debris, and industry. Such mini vehicles require smart navigation for efficiently identifying the object it is searching for. However, currently, the autonomous driving of such low-scale mini vehicles is a difficult task. TinyML approach is investigated to seek for improvement of autonomous driving of such mini vehicles (de

Prado et al., 2021). It is deployed on the GAP8 MCI based on the convolution neural network framework. It is also tested on the STM32L4 and NXP k64f platforms. Results show that the integration minimizes processing latency by 13x and improves energy consumption by 92%. Fig. 39. presents the closed-loop learning of the mini vehicle. It consumes only 3.9 μJ per inference.

Automated traffic scheduling is recently investigated for possible integration with TinyML framework to improve the real-time traffic system (Roshan et al., 2021). The implied method uses piezoelectric sensors embedded in many lanes of a road. A two-point time ratio method is sought to detect a vehicle by using the piezo-sensors' data. The vehicles are then classified for prediction of green light timings. The study uses a random forest regressor (RFR) to predict the duration of the signal based on the input vehicle count in each lane. It is deployed on an Arduino Uno, augmented with the m2gen library with support from the Scikit Learn. The deployed algorithm needs only 1.754 KB.

5.15. Anomaly detection

Anomaly refers to some point value that is different from the majority mass of the points. In (Mansoureh Lord, 2021), an investigation is made to seek for the appropriateness of TinyML for anomaly detection related tasks. It uses a generic artificial neural network, autoencoder, and variational autoencoder on the Arduino Nano 33 BLE sense module. The study uses top-load washing machine Kenmore model for detecting anomalies in the unbalanced spin dry cycle. The results show around 92% accuracy and 90% precision.

6. Prospects

In this section, we discuss various key challenges associated with TinyML and a future road map.

6.1. Open challenges

- *Low Power:* Major research issue for TinyML is the minimal power availability in the edge devices. TinyML must accommodate the energy efficiency feature in its paradigm. TinyML models require a certain level of energy to maintain the accuracy level of the algorithms. It is difficult to specify a universal power-management module for hardware platforms due to the variation in their designs and pre-processing paths. Edge devices are normally connected to sensors and other peripherals that may also cause some power aware mismanagement in TinyML systems. Thus, energy efficient TinyML system design remains a most critical challenge.
- *Limited Memory:* It is another vertical of resistance that hinders the growth of TinyML. Due to the extremely small (e.g., KB) size SRAM and less than 1 MB size flash memory makes the task of machine learning very difficult to get deployed at the time edge devices. Traditional machine learning inference can use higher peaks of memory (e.g., GB in size) which is not possible for edge platforms. TinyML aware devices can't spare its cores for system-under-test (SUT) mechanisms. Thus, low memory foot-print of edge hardware makes the TinyML systems emerge.
- *Hardware and Software Heterogeneity:* Heterogeneity of hardware and software infrastructure pose a great challenge for TinyML systems to adopt a given method of learning and deployment strategy. The SUT doesn't ordinarily include generic features (e.g., system clock) in TinyML settings. Moreover, normalizing the edge aware performance results can't be made homogeneous due to the form factor of various hardware platforms. TinyML needs to follow code handling, code generation, and interpretation of machine learning models in order to leverage acceptable accuracy rates. Furthermore, currently used software frameworks are not all alike. We envisage that software heterogeneity should be resolved to allow TinyML to be feasibly deployed in resource frugal devices.
- *Lack of Benchmarking Tools:* Conventional machine learning benchmarks need inference models. Such inference models consume huge energy during the peak processing moment. They also consume a huge amount of memory space in existing hardware. In the TinyML scenario, it is difficult to accommodate due to lack of memory and processing capability. Even if one attempts to deploy an individual benchmark suitable for a TinyML setting, one must be aware of huge power consumption to run the benchmark. We expect that such benchmarks should allow multi-level of quantization and precision while covering the diversity of the target application. Existing benchmarks must be re-engineered before applying to TinyML systems so that enough flexibility is attained to maintain hardware heterogeneity. We expect that such benchmarks should be designed with proper balance with optimal aspect, representativeness, and portability features. The benchmark should be aligned with machine learning interpreters that can provide model independent behavior (e.g., TFLM) (Banbury et al., 2020). More initiatives should be taken to provide benchmarks as currently served by the TinyMLPerf organization.
- *Lack of Datasets:* Existing datasets may not be suitable for TinyML architecture due to the lack of adaptability in terms of low-power consuming perspective. Such datasets should possess appropriate temporal and spatial resolution to correspond to the feature of data generation from external sensors. Also, the noise level and diversity of such a dataset must be associated with the low-power edge device aware facilitation. Thus, we emphasize the need for a standard set of datasets that can be readily considered for training the TinyML systems.
- *Lack of Popularly Accepted Models:* Currently, many machine learning models are widely accepted for conventional infrastructure. For example, MobileNet is used as the baseline for benchmarking of deep neural networks in mobile edge computing devices. However, no such popular model is still not in use that can be adopted for the TinyML ecosystem. We envisage that such models shall soon be floated in the research domain to improve the employability of TinyML paradigm.
- *Edge Computing Infrastructure:* Existing edge computing domain is in a nascent stage that resists the adoption of dynamically changing resources in the edge devices. Further, the edge configuration is not properly supported by currently deployed edge infrastructure. Key issues persist that involve device mobility and reliability features during the machine learning model deployments. We can foresee the possible enablers that can support the edge computing infrastructure for adopting TinyML. For example, container technology, network virtualization, isomorphic techniques, and handover schemes can alleviate the adoption of TInyML in existing edge infrastructure.
- *Edge Platform Orchestration:* Present edge platforms face resource allocation problems. Moreover, resource distribution and dynamic resource allocation are also put on top of current issues. Several data analysis algorithms and data intelligent techniques can resolve the edge platform orchestration towards a fully distributed spectrum. We emphasize using virtualized optimization schemes to support multi-level dynamics at the edge.
- *Data and Network Management:* Deploying TinyML is not enough when the data and network is not properly managed. We find that the current edge network lacks understating of heterogeneous data types causing a challenge for less intelligent

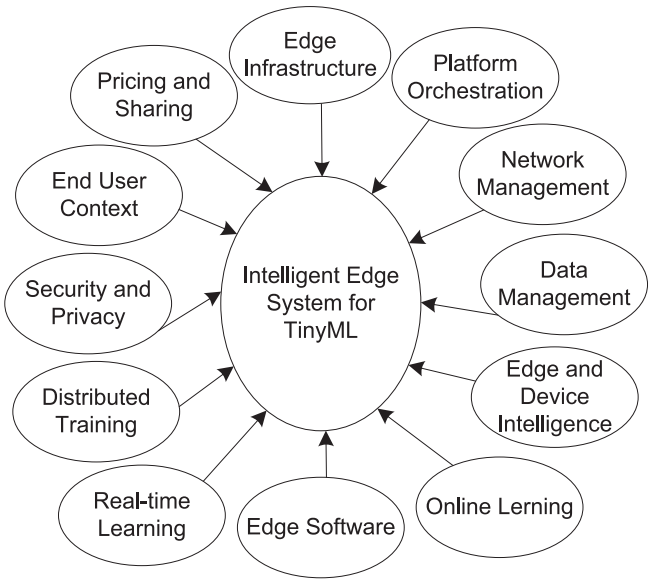

Fig 40. Key enablers of TinyML.

performance. It is integrated with lack of understanding about data life cycle for sensor equipped devices. Inconsistent sensory data structure alleviates the problem of edge data management. Several solutions can be prescribed to address this issue. For example, extension of data and sensor fusion algorithms can be used for network management at the edge. One can investigate the significance of network fault detection and identification thereto. Incremental learning algorithms should be fused with knowledge extraction strategies to improve the knowledge sharing possibilities. Light-weight machine learning solutions can be considered to increase the autonomy and self-adaptive capabilities at the edge network management.

- *Reliability:* It is an important metric of adoption of TinyML systems for large-scale real-life use cases (Shafique et al., 2021). Major challenges of reliability at edge devices are observed as follows, (i) process variations, (ii) soft errors, and (iii) aging. Process variations take place during the imprecision in the chip fabrication phases. Power leakage and degradation of wafer in a chip can alleviate the process variation scenario in edge devices. High energy particles can cause the tiny edge devices to deviate from its accuracy causing soft errors. Edge devices may be positioned in remote locations with minimal human intervention. Such placement may cause the edge device harm from natural phenomena to degrade the processing capacity of the circuit, on-board battery life, and real-time clock circuits. It is important to ascertain that an edge device per its reliability assessment before deploying them in the field of applications.
- *Software Development for Edge:* Designing software for edge needs specialized skills sets such as alignment of dynamic configurations of edge devices, assessing security issues, and flexible deployment constraints. Once again, the container technologies can be of help to develop DevOps with tuning from virtualization aspects. One can think of designing liquid software for propagation from one edge node to another to help machine learning models. Intersection of edge device and edge software is another key challenge. It can be resolved by using light-weight machine learning solutions to increase self-capabilities of the edge node. Various agents can be deployed with in-built cognitive capabilities for learning and proactive behavior. Advanced machine learning techniques should be in place to enable cooperation, collaboration, and adoption with game theory, swarm intelligence, and genetic algorithms.
- *Need of New Machine Learning Models:* This is high time that we need new machine learning models for the TinyML ecosystem. Such models should be able to provide short-time responses. We can use federated learning, transfer learning, reinforcement learning, and online learning while aggregating with knowledge distillation dimension. Machine learning models should be codesigned with control and communication aspects to provide real-time solutions. Model pruning and quantization should be the inevitable parts of such a model design process. It should be warranted that overall cost reduction is done while coordination with edge devices is made.

6.2. Future road map

- *New Dimension:* We emphasize the future growth and adoption of the TinyML paradigm shall depend on several aspects. For example, an intelligent edge system requires the edge software to integrate online learning, real-time learning, distributed training, coherence between edge-device intelligence, sophisticated and data-network management. Such cooperation should be extended for localized security and privacy improvement so that end-user context can be preserved at edge devices. Furthermore, we should focus on edge infrastructure development, edge platform orchestration, and low-cost knowledge sharing capabilities into the edge systems. Fig. 40. shows the contributions of such parameters for growth of TinyML paradigm (, xxxx).
- *Edge Inteligence Framework:* Typical edge intelligence framework should depend on following, such as, (i) wireless networking, data intelligence, (iii) cooperative intelligence, (iv) energy efficient management, (v) dynamic task allocation, (vi) liquid software propagation among edge nodes, (vii) implementation of communication services, (viii) real-time inference, (ix) machine learning-as-a service, and (x) provisioning of predictive quality-of-service. One can think of integrating such TinyML settings as a new generation learning perspective, thus making them adopted with the 5G and 6G technologies. Doing

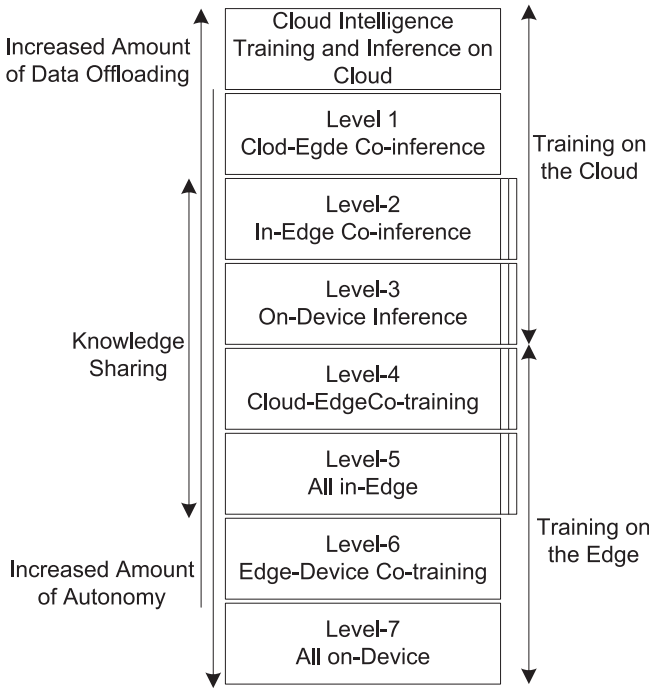

Fig. 41. Level rating of edge intelligence.

so, TinyML systems should be enabled with mmWave xhaul systems where optimization of machine learning models can be augmented. Hyper collaboration between cloud and edge can be then realized in an efficient manner.

- *Task Offloading:* On/off-loading of computational tasks can be actually harnessed during edge enabled machine learning scenarios. Such loading strategies should be an add-on to existing dynamic configuration of edge-aware specifics. Doing so, TinyML can empower the transfer of resource-intensive jobs from the resource frugal edge devices. Thus, an interoperability between edge and cloud can be better oriented. Investigation should be paved to seek for the underlying processes behind such resource allocations (e.g., CPU cycles, memory chunks, channel bandwidth, sensing capabilities, data hiring, machine learning model sharing).
- *Mobility Support:* Mobility support is another avenue that can be resolved by deploying improved liquid computing handover schemes in the edge platforms. Location-aware optimization can rescue the network bandwidth and increase the coverage of network spectrum for benefiting edge devices to establish cooperation among nearby devices. Doing so can help to progressively transfer knowledge and models for reuse by others. Furthermore, quality-of-service can be modified to predict the behavior of edge devices while interacting with neighbor nodes. Such information can be shared among the near vicinity nodes or clouds to predict the intelligent-aware orientation of the edge devices.
- *Level Rating:* In (Zhou et al., 2019), the edge intelligence is categorized into 6 levels. The levels are classified based on the amount of data offloading and the associated path length thereto. The diagram depicts the edge intelligence in terms of a cluster or a federated edge node or an intelligent node or a single node or an autonomous node. Cloud intelligence is the top most level that provides training and inferencing capability at the cloud only. Level-1 allows to train the machine learning model at cloud, but inferencing is done at the edge in an edge-cloud collaborative manner (partial offloading of data to cloud). Level-2 behaves as of Level-1 where the inference is done in edge only as an independent manner or partial coordinated manner. Level-3 separates training at cloud from inferencing at edge devices with no need of offloading of data. Level-4 allows both training and inference at edge devices with cooperation from the cloud. Level-5 deals with training and inference of all in-edge devices. Level-6 states the training and inferencing at edge device with cooperation from edge device only. Level-7 allows to train and infer the machine learning model in the on-device manner. Fig. 41. presents a 7-level rating of edge intelligence (White Paper on Edge Intelligence, 2021).
- *Future Insight:* TinyML is gradually becoming a need and reality for solving decision aware real-life issues. Especially, for low-power embedded device pools that are being used for various applications. Wearable platforms need to accommodate the TinyML orientation to allow users towards sophisticated use-case experience. We suggest the techno-developers and enterprises to incorporate the TinyML aware solutions for future generation smart and handheld devices. There is a strong need of minimization of resource hungry CPU-GPU-TPU interaction for facilitation of smart decision-making supports. Microcontroller manufacturers should emphasize on the integrated TinyML design specifications so that users need not to bother about external alignments related to AI. We should aim to integrated TinyML into the IoT-edge analytics domain for making the application more user friendly and reliable. Standardized process flow should be developed to help developers for realization of the ready-to-market deployment scenario. Proper dataset repository and light-weight benchmarking tools should be developed. Industry X.0 applications should be targeted with TinyML accommodation in the coming days. Moreover, 8-bit microcontroller platforms should be leveraged with such low-memory foot print libraries. Delay mitigation at the edge-level implications should be resolved by using TinyML. We emphasize the use of TinyML in the low-cost and hand-held digital devices to provide instantaneous feedback to the users. Unnecessary usage and dependency on the GPUs, TPUs, and cloud platforms can be minimized by proper utilization of TinyML frameworks.

7. Conclusion

Edge-IoT ecosystem has tremendous possibility to leverage smart decision-making capability at the edge of the network. Involving machine learning in tiny resource limited embedded devices is becoming an important need in view of futuristic applications. TinyML paradigm needs to investigated in-detail to upgrade current edge-aware machine learning context. In this article presents intuitive detailing about TinyML covering various dimensions of research. We envisage that TinyML paradigm shift is certainly possible with due detailing in the in-memory processing and neuromorphic computing. We also prescribe that existing benchmark should be revisited with new datasets for allowing them to integrate with low-power embedded devices. We expect that low-footprint machine learning generating translator should be developed to enable edge-IoT tool chain. We suggest the enterprises and enthusiasts to come forward in this direction and work towards development of new benchmarks, new optimized datasets, prepare TinyML process flows, and incorporate TinyML into the day-to-day low-cost and low-power smart hand-held devices including smart phone, microcontrollers, and IoT-edge systems. TinyML has tremendous possibility to change existing decision-making perspective to solve larger problems in smaller chunks that too at the resource constrained edge of network.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Agora Product Development Kit, https://os.mbed.com/platforms/AGORA-DEV/.
AI-deck 1.1, https://store.bitcraze.io/products/ai-deck-1-1.
Alongi, F., Ghielmetti, N., Pau, D., Terraneo, F. and Fornaciari, W., 2020, September. Tiny Neural Networks for Environmental Predictions: an integrated approach with Miosix. In 2020 IEEE International Conference on Smart Computing (SMARTCOMP) (pp. 350-355). IEEE.
Alwarafy, A., Al-Thelaya, K. A., Abdallah, M., Schneider, J., Hamdi, M., 2021, "A Survey on Security and Privacy Issues in Edge-Computing-Assisted Internet of Things," in: IEEE Internet of Things Journal, vol. 8, no. 6, pp. 4004-4022, doi: 10.1109/JIOT.2020.3015432.
Amber: A Complete, ML-Based, Anomaly Detection Pipeline for Microcontrollers, https://cms.tinyml.org/wp-content/uploads/talks2020/tinyML_Talks_Brian_Turnquist_and_Rodney_Dockter_201124.pdf, Accessed on November, 2021.
Anusuya, R., Renuka, D. K., Kumar, L. A., 2021, "Review on Challenges of Secure Data Analytics in Edge Computing," in: 2021 International Conference on Computer Communication and Informatics (ICCCI), pp. 1-5, doi: 10.1109/ICCCI50826.2021.9402559.
Apollo3, https://www.sparkfun.com/products/15170
Arduino Nano 33 BLE Sense, https://store-usa.arduino.cc/products/arduino-nano-33-ble-sense,
Arduino Portenta H7, https://store.arduino.cc/products/portenta-h7.
ARM-TinyL, https://www.arm.com/blogs/blueprint/tinyml, Accessed on November, 2021.
Artificial Neural Networks and Tools for Microcontrollers, https://sites.google.com/g.harvard.edu/tinyml/lectures?authuser=0#h.ocndbasg81yl, Accessed on November, 2021.
Banbury, C.R., Reddi, V.J., Lam, M., Fu, W., Fazel, A., Holleman, J., Huang, X., Hurtado, R., Kanter, D., Lokhmotov, A. and Patterson, D., 2020. Benchmarking TinyML systems: Challenges and direction. arXiv preprint arXiv:2003.04821.

Banbury, C., Reddi, V.J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., Montino, P., Kanter, D., Ahmed, S., Pau, D. and Thakker, U., 2021. MLPerf Tiny Benchmark. arXiv preprint arXiv:2106.07597.

Bao, W., Wu, C., Guleng, S., Zhang, J., Yau, K.-L.A., Ji, Y., 2021. Edge computing-based joint client selection and networking scheme for federated learning in vehicular IoT. China Commun. 18 (6), 39–52. https://doi.org/10.23919/JCC.2021.06.004.

Basaklar, T., Tuncel, Y., Narayana, S.Y., Gumussoy, S. and Ogras, U.Y., 2021. Hypervector Design for Efficient Hyperdimensional Computing on Edge Devices. arXiv preprint arXiv:2103.06709.

Benchmark-NN, https://github.com/bharathsudharsan/TinyML-Benchmark-NNs-on-MCUs

Benmeziane, H., El Maghraoui, K., Ouarnoughi, H., Niar, S., Wistuba, M., Wang, N., Burdick, D. and Lewis, D., 2021, August. Hardware-Aware Neural Architecture Search: Survey and Taxonomy. In Thirtieth International Joint Conference on Artificial Intelligence {IJCAI-21} (pp. 4322-4329). International Joint Conferences on Artificial Intelligence Organization.

Bian, S., Lukowicz, P., 2021. September. Capacitive Sensing Based On-board Hand Gesture Recognition with TinyML. In: Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers, pp. 4–5.

Bringmann, O., Ecker, W., Feldner, I., Frischknecht, A., Gerum, C., Hämäläinen, T., Hanif, M.A., Klaiber, M.J., Mueller-Gritschneder, D., Bernardo, P.P., Prebeck, S., 2021. September. Automated HW/SW co-design for edge AI: state, challenges and steps ahead. In: Proceedings of the 2021 International Conference on Hardware/Software Codesign and System Synthesis, pp. 11–20.

Cai, H., Gan, C., Wang, T., Zhang, Z. and Han, S., 2019. Once-for-all: Train one network and specialize it for efficient deployment. arXiv preprint arXiv:1908.09791.

Capotondi, A., Rusci, M., Fariselli, M., Benini, L., 2020. Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices. IEEE Trans. Circ. Syst. II: Express Briefs 67 (5), 871–875.

CC1352P Launchpad, https://www.ti.com/tool/LAUNCHXL-CC1352P.

Chai, S.M., 2020, December. Quantization-Guided Training for Compact TinyML Models. In Research Symposium on Tiny Machine Learning.

Cough detection, https://create.arduino.cc/projecthub/edge-impulse/cough-detection-with-tinyml-on-arduino-417f37

Curnick, D.J., Davies, A.J., Duncan, C., Freeman, R., Jacoby, D.M., Shelley, H.T., Rossi, C., Wearn, O.R., Williamson, M.J., Pettorelli, N., 2021. SmallSats: a new technological frontier in ecology and conservation? Remote Sens. Ecol. Conserv.

Data Collection Design for Real World TinyML, https://sites.google.com/g.harvard.edu/tinyml/lectures?authuser=0#h.5aj7gww1ta6s, Accessed on November, 2021.

de Prado, M., Rusci, M., Capotondi, A., Donze, R., Benini, L., Pazos, N., 2021. Robustifying the Deployment of tinyML Models for Autonomous mini-vehicles. Sensors 21 (4), 1339.

Ding, C., Zhou, A., Ma, X., Zhang, N., Hsu, C. -H., Wang, S., "Towards Diversified IoT Services in Mobile Edge Computing," in: IEEE Transactions on Cloud Computing, doi: 10.1109/TCC.2021.3109385.

Disabato, S., Roveri, M., 2020. November. Incremental on-device tiny machine learning. In: Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things, pp. 7–13.

Dogaru, R., Dogaru, I., 2020. Fast Training of Light Binary Convolutional Neural Networks using Chainer and Cupy. In: 2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). IEEE, pp. 1–4.

Doyu, H., Morabito, R., Höller, J., 2020. Bringing machine learning to the deepest IoT edge with TinyML as-a-service. IEEE IoT Newsl.

ECM3532 AI Sensor Neuro sensor processor (NSP), https://www.fierceelectronics.com/electronics/multicore-processor-brings-ai-to-sensing-apps#:~:text=The%20ECM3532%20is%20a%20Neural,microwatts%20for%20many%20sensing%20applications.

Edge Impulse, https://www.edgeimpulse.com/

EdgeML: Algorithms for TinyML, https://sites.google.com/g.harvard.edu/tinyml/lectures?authuser=0#h.5hc2tcel4ikp, Accessed on November, 2021.

ELL, https://microsoft.github.io/ELL/

Endpoint AI and the Advent of the microNPU, https://sites.google.com/g.harvard.edu/tinyml/lectures?authuser=0#h.eefoswaclfo9, Accessed on November, 2021.

ESP-EYE, https://www.espressif.com/en/products/devkits/esp-eye/overview.

Estrebou, C.A., Fleming, M., Saavedra, M.D., Adra, F., 2021. MbedML: A Machine Learning Project for Embedded Systems. In IX Jornadas de Cloud Computing. Big Data Emerging Topics (Modalidad virtual 22. al 25 de junio de 2021).

Fahim, F., Hawks, B., Herwig, C., Hirschauer, J., Jindariani, S., Tran, N., Carloni, L.P., Di Guglielmo, G., Harris, P., Krupa, J. and Rankin, D., 2021. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. arXiv preprint arXiv:2103.05579.

Fatemi, H., Karia, V., Pandit, T. and Kudithipudi, D., 2020, December. TENT: Efficient Quantization of Neural Networks on the tiny Edge with Tapered FixEd PoiNT. In Research Symposium on Tiny Machine Learning.

Forbes-TinyML, https://www.forbes.com/sites/janakirammsv/2020/11/03/how-tinyml-makes-artificial-intelligence-ubiquitous/, Accessed on November, 2021.

FRDM-K64F, https://www.nxp.com/design/development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F.

GAP8, https://greenwaves-technologies.com/wp-content/uploads/2021/04/Product-Brief-GAP8-V1_9.pdf

GAP9, https://greenwaves-technologies.com/gap9_iot_application_processor/.

Ge, L., Parhi, K.K., 2020. Classification using hyperdimensional computing: a review. IEEE Circ. Syst. Magazine 20 (2), 30–47.

Giordano, M., Mayer, P. and Magno, M., 2020, November. A Battery-Free Long-Range Wireless Smart Camera for Face Detection. In Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems (pp. 29-35).

Gopinath, S., Ghanathe, N., Seshadri, V., Sharma, R., 2019. June. Compiling kb-sized machine learning models to tiny iot devices. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 79–95.

Goudarzi, M., Palaniswami, M. S., Buyya, R., "A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments," in IEEE Transactions on Mobile Computing, doi: 10.1109/TMC.2021.3123165.

Grau, M.M., Centelles, R.P. and Freitag, F., 2021, September. On-Device Training of Machine Learning Models on Microcontrollers With a Look at Federated Learning. In Proceedings of the Conference on Information Technology for Social Good (pp. 198-203).

Guleria, C., Das, K., Sahu, A., 2021, "A Survey on Mobile Edge Computing: Efficient Energy Management System," in: 2021 Innovations in Energy Management and Renewable Resources(52042), pp. 1-4, doi: 10.1109/IEMRE52042.2021.9386951.

Hardy, E. and Badets, F., 2021. An Ultra-low Power RNN Classifier for Always-On Voice Wake-Up Detection Robust to Real-World Scenarios. arXiv preprint arXiv:2103.04792.

Himax EW-I Plus, https://www.sparkfun.com/products/17256.

Hu, P., Im, J., Asgar, Z., Katti, S., 2020. November. Starfish: resilient image compression for AIoT cameras. In: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, pp. 395–408.

Jiao, B., Zhang, J., Xie, Y., Wang, S., Zhu, H., Kang, X., Dong, Z., Zhang, L. and Chen, C., 2021, January. A 0.57-GOPS/DSP Object Detection PIM Accelerator on FPGA. In 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC) (pp. 13-14). IEEE.

Kopparapu, K. and Lin, E., 2021. TinyFedTL: Federated Transfer Learning on Tiny Devices. arXiv preprint arXiv:2110.01107.

Kwon, J., Park, D., 2021. January. Toward Data-Adaptable TinyML using Model Partial Replacement for Resource Frugal Edge Device. In: The International Conference on High Performance Computing in Asia-Pacific Region, pp. 133–135.

Applied Machine Learning for Embedded IoT Devices, https://sites.google.com/g.harvard.edu/tinyml/home, Accessed on November, 2021.

Li, B., Chen, P., Liu, H., Guo, W., Cao, X., Du, J., Zhao, C., Zhang, J., 2021a. Random sketch learning for deep neural networks in edge computing. Nature Comput. Sci. 1 (3), 221–228.

Li, W., Deng, W., She, R., Zhang, N., Wang, Y., Ma, W., 2021, "Edge Computing Offloading Strategy Based on Particle Swarm Algorithm for Power Internet of Things," in: 2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), pp. 145-150, doi: 10.1109/ICBAIE52039.2021.9389919.

Li, S., Romaszkan, W., Graening, A. and Gupta, P., 2021. SWIS–Shared Weight bIt Sparsity for Efficient Neural Network Acceleration. arXiv preprint arXiv:2103.01308.

Li, Z., Gao, J., Lai, J., 2021d. HBDCA: A Toolchain for High-Accuracy BRAM-Defined CNN Accelerator on FPGA with Flexible Structure. IEICE Trans. Inf. Syst. 104 (10), 1724–1733.

Liberis, E., Dudziak, Ł., Lane, N.D., 2021. April. μNAS: Constrained Neural Architecture Search for Microcontrollers. In: Proceedings of the 1st Workshop on Machine Learning and Systems, pp. 70–79.

Liu, J., Liu, C., Wang, B., Gao, G., Wang, S., "Optimized Task Allocation for IoT Application in Mobile Edge Computing," in: IEEE Internet of Things Journal, doi: 10.1109/JIOT.2021.3091599.

Lu, C. -H., Lin, X. -Z., 2021, "Toward Direct Edge-to-Edge Transfer Learning for IoT-Enabled Edge Cameras," in: IEEE Internet of Things Journal, 8(6), pp. 4931-4943, doi: 10.1109/JIOT.2020.3034153.

Luukkonen, T., Colley, A., Seppänen, T. and Häkkilä, J., 2021, February. Cough Activated Dynamic Face Visor. In Augmented Humans Conference 2021 (pp. 295-297).

Mansoureh Lord, TinyML Anomaly Detection, Thesis, https://scholarworks.csun.edu/bitstream/handle/10211.3/219966/Lord-Mansoureh-thesis-2021.pdf?sequence=1

Mathur, A., Beutel, D.J., de Gusmão, P.P.B., Fernandez-Marques, J., Topal, T., Qiu, X., Parcollet, T., Gao, Y. and Lane, N.D., 2021. On-device Federated Learning with Flower. arXiv preprint arXiv:2104.03042.

Mazumder, M., Banbury, C., Meyer, J., Warden, P. and Reddi, V.J., 2021. Few-Shot Keyword Spotting in Any Language. arXiv preprint arXiv:2104.01454.

Miao, H. and Lin, F.X., 2021. Enabling Large Neural Networks on Tiny Microcontrollers with Swapping. arXiv preprint arXiv:2101.08744.

MKR Video 4000, https://store.arduino.cc/products/arduino-mkr-vidor-4000.

MLOps for TinyML, https://sites.google.com/g.harvard.edu/tinyml/lectures?authuser=0#h.m9uxfxjs8d5u, Accessed on November, 2021.

Recent Progress on TinyML Technologies and Opportunities, https://sites.google.com/g.harvard.edu/tinyml/lectures?authuser=0#h.839rbio9569w, Accessed on November, 2021.

Rashid, H.A., Ren, H., Mazumder, A.N. and Mohsenin, T., 2020. Tiny RespNet: A Scalable Multimodal TinyCNN Processor for Automatic Detection of Respiratory Symptoms.

MLPerfTiny v.5, https://mlcommons.org/en/news/mlperf-tiny-v05/

Mohan, P., Paul, A.J. and Chirania, A., 2021. A tiny CNN architecture for medical face mask detection for resource-constrained endpoints. In Innovations in Electrical and Electronic Engineering (pp. 657-670). Springer, Singapore.
Muhammad, G., Hossain, M. S., "Emotion Recognition for Cognitive Edge Computing Using Deep Learning," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2021.3058587.
Muniswamaiah, M., Agerwala, T., Tappert, C.C., 2021. A Survey on Cloudlets Mobile Edge, and Fog Computing. In: 2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 139–142. https://doi.org/10.1109/CSCloud-EdgeCom52276.2021.00034.
Nakhle, F., Harfouche, A.L., 2021. Ready, Steady, Go AI: A practical tutorial on fundamentals of artificial intelligence and its applications in phenomics image analysis. Patterns 2 (9).
NanoEdge AI Studio, https://cartesiam-neai-docs.readthedocs-hosted.com/
Enabling Neural network at the low power edge: A neural network compiler for hardware constrained embedded system, https://cms.tinyml.org/wp-content/uploads/talks2020/tinyML_Talks_Chao_Xu_201124.pdf, Accessed on November, 2021.
Nezami, Z., Zamanifar, K., Djemame, K., Pournaras, E., 2021. Decentralized edge-to-cloud load balancing: service placement for the internet of things. IEEE Access 9, 64983–65000. https://doi.org/10.1109/ACCESS.2021.3074962.
Nicla Sense ME, https://docs.arduino.cc/hardware/nicla-sense-me/.
Nordic Semi nRF52840 DK, https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk.
Nordic Semi Thingy:91, https://www.nordicsemi.com/Products/Development-hardware/Nordic-Thingy-91.
Nvidia Jetson Nano, https://developer.nvidia.com/embedded/jetson-nano-developer-kit.
Ogino, T., 2021. Simplified Multi-objective Optimization for Flexible IoT Edge Computing. In: 2021 4th International Conference on Information and Computer Technologies (ICICT), pp. 168–173. https://doi.org/10.1109/ICICT52872.2021.00035.
OpenMV Cam H7 Plus, https://openmv.io/products/openmv-cam-h7-plus.
Paissan, F., Ancilotto, A. and Farella, E., 2021. PhiNets: a scalable backbone for low-power AI at the edge. arXiv preprint arXiv:2110.00337.
uTVM, https://octoml.ai/blog/tinyml-tvm-taming-the-final-ml-frontier
Paul, A.J., Mohan, P. and Sehgal, S., 2020, December. Rethinking generalization in american sign language prediction for edge devices with extremely low memory footprint. In 2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS) (pp. 147-152). IEEE.
Pico4ML BLE, https://www.arducam.com/docs/pico/arducam-pico4mltinymldevkit/.
Privacy in Context, https://sites.google.com/g.harvard.edu/tinyml/lectures?authuser=0#h.gy760llziipy, Accessed on November, 2021
PyTorch, https://pytorch.org/
Raspberry Pi 4B, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.
Ren, H., Anicic, D. and Runkler, T., 2021. TinyOL: TinyML with Online-Learning on Microcontrollers. arXiv preprint arXiv:2103.08295.Tabanelli, E., Tagliavini, G. and Benini, L., 2021. DNN is not all you need: Parallelizing Non-Neural ML Algorithms on Ultra-Low-Power IoT Processors. arXiv preprint arXiv:2107.09448.
Ren, W., Sun, Y., Luo, H., Guizani, M., 2021, "A Demand-driven Incremental Deployment Strategy for Edge Computing in IoT network," in: IEEE Transactions on Network Science and Engineering, doi: 10.1109/TNSE.2021.3120270.
Ren, H., Anicic, D. and Runkler, T., 2021. The synergy of complex event processing and tiny machine learning in industrial IoT. arXiv preprint arXiv:2105.03371.
Roshan, A.N., Gokulapriyan, B., Siddarth, C. and Kokil, P., 2021, March. Adaptive Traffic Control With TinyML. In 2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET) (pp. 451-455). IEEE.
Rusci, M., Fariselli, M., Capotondi, A., Benini, L., 2020. Leveraging Automated Mixed-Low-Precision Quantization for Tiny Edge Microcontrollers. In: IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning. Springer, Cham, pp. 296–308.
Sanchez-Iborra, R., Skarmeta, A.F., 2020. Tinyml-enabled frugal smart objects: Challenges and opportunities. IEEE Circ. Syst. Magazine 20 (3), 4–18.
Seeed Wio Terminal, https://www.seeedstudio.com/Wio-Terminal-p-4509.html.
Shafique, M., Marchisio, A., Putra, R.V.W. and Hanif, M.A., 2021. Towards Energy-Efficient and Secure Edge AI: A Cross-Layer Framework. arXiv preprint arXiv:2109.09829.
Signoretti, G., Silva, M., Andrade, P., Silva, I., Sisinni, E., Ferrari, P., 2021. An Evolving TinyML Compression Algorithm for IoT Environments Based on Data Eccentricity. Sensors 21 (12), 4153.
Singh, J., Bello, Y., Hussein, A. R., Erbad, A., Mohamed, A., 2021, "Hierarchical Security Paradigm for IoT Multiaccess Edge Computing," in: IEEE Internet of Things Journal, vol. 8, no. 7, pp. 5794-5805, 1 April1, 2021, doi: 10.1109/JIOT.2020.3033265.
Sony's Spresense TinyML Board, https://developer.sony.com/develop/spresense/.
ST IoT Discovery, https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html.
STM32Cube.AI, https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html
STM32F Discovery, https://www.st.com/en/evaluation-tools/stm32f4discovery.html
Strubell, E., Ganesh, A. and McCallum, A., 2019. Energy and policy considerations for deep learning in NLP. arXiv preprint arXiv:1906.02243.
Sudharsan, B., Salerno, S., Nguyen, D.D., Yahya, M., Wahid, A., Yadav, P., Breslin, J.G. and Ali, M.I., 2021. TinyML benchmark: Executing fully connected neural networks on commodity microcontrollers. In IEEE 7th World Forum on Internet of Things (WF-IoT), New Orleans, Louisiana, USA.
Svoboda, F., Nunes, D., Alizadeh, M., Daries, R., Luo, R., Mathur, A., Bhattacharya, S., Silva, J.S., Lane, N.D., 2020. Resource Efficient Deep Reinforcement Learning for Acutely Constrained TinyML Devices. Research Symposium on Tiny Machine Learning.
Tabanelli, E., Tagliavini, G. and Benini, L., 2021. DNN is not all you need: Parallelizing Non-Neural ML Algorithms on Ultra-Low-Power IoT Processors. arXiv preprint arXiv:2107.09448.
TensorFlow Lite (TFL), https://www.tensorflow.org/lite
Thakker, U., Whatmough, P.N., Liu, Z.G., Mattina, M. and Beu, J., 2020. Compressing language models using doped kronecker products. arXiv preprint arXiv:2001.08896.
Thunderboard Sense 2, https://www.silabs.com/development-tools/thunderboard/thunderboard-sense-two-kit.
TinyML as-a-Service, https://www.ericsson.com/en/blog/2019/12/tinyml-as-a-service
TinyML, https://cms.tinyml.org/wp-content/uploads/emea2021/tinyML_Talks_Felix_Johnny_Thomasmathibalan_and_Fredrik_Knutsson_210208.pdf, Accessed on November, 2021.
TinyML, Available Online https://www.tinyml.org/, Accessed on November, 2021.
TinyMLPerf, https://github.com/mlcommons/tiny
Unsupervised collaborative learning technology at the Edge, https://cms.tinyml.org/wp-content/uploads/talks2020/tinyML_Talks_Alexander_Eroma_200428.pdf, Accessed on November, 2021.
uTensor, http://utensor.ai
uTVM system, https://tvm.apache.org/2020/06/04/tinyml-how-tvm-is-taming-tiny
Vuletic, M., Mujagic, V., Milojevic, N. and Biswas, D., Edge AI Framework for Healthcare Applications.
Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S., 2019. Haq: Hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8612–8620.
Wang, X., Magno, M., Cavigelli, L., Benini, L., 2020. 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. 7 (5), 4403–4417.
Warden, P. and Situnayake, D., 2019. Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers. O'Reilly Media.
6G White Paper on Edge Intelligence, https://arxiv.org/abs/2004.14850.
Wong, A., Famouri, M., Pavlova, M. and Surana, S., 2020. Tinyspeech: Attention condensers for deep speech recognition neural networks on edge devices. arXiv preprint arXiv:2008.04245.
Wong, A., Famouri, M. and Shafiee, M.J., 2020. AttendNets: Tiny Deep Image Recognition Neural Networks for the Edge via Visual Attention Condensers. arXiv preprint arXiv:2009.14385.
Wu, D., Huang, X., Xie, X., Nie, X., Bao, L., Qin, Z., 2021, "LEDGE: Leveraging Edge Computing for Resilient Access Management of Mobile IoT," in: IEEE Transactions on Mobile Computing, vol. 20, no. 3, pp. 1110-1125, doi: 10.1109/TMC.2019.2954872.
XCore.ai, https://www.xmos.ai/xcore-ai/.
Ying, J. et al., 2021. Edge-enabled cloud computing management platform for smart manufacturing. In: 2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), pp. 682–686. https://doi.org/10.1109/MetroInd4.0IoT51437.2021.9488441.
Yoo, J., Lee, D., Son, C., Jung, S., Yoo, B., Choi, C., Han, J.J., Han, B., 2021. RaScaNet: Learning Tiny Models by Raster-Scanning Images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 13673–13682.
Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., Zhang, J., 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proc. IEEE 107 (8), 1738–1762.
Zhou, A., Muller, R. and Rabaey, J., 2021. Memory-Efficient, Limb Position-Aware Hand Gesture Recognition using Hyperdimensional Computing. arXiv preprint arXiv:2103.05267.
Ziaul Haque Zim, M., 2021. TinyML: Analysis of Xtensa LX6 microprocessor for Neural Network Applications by ESP32 SoC. arXiv e-prints, pp.arXiv-2106.