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

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.

- Previous article in issue
- Next article in issue

Keywords

TinyML

IoT

Edge intelligence

Energy efficient AI

Resource constrained intelligence

Embedded AI

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, 2021b, TinyML, 2021a).

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:

- 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.

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, 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

. Download: Download high-res image (186KB)

. Download: Download full-size image

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

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 real 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.

. Download: Download high-res image (237KB)

. Download: Download full-size image

Fig. 2. TinyML has miniscule scalability.

. Download: Download high-res image (218KB)

. Download: Download full-size image

Fig. 3. Layered approach with respect to TinyML.

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 RTOS, 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).

. Download: Download high-res image (182KB)

. Download: Download full-size image

Fig. 4. Composition of TinyML

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 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.

. Download: Download high-res image (139KB)

. Download: Download full-size image

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

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.

Table 1. Comparison Among Hardware Platforms to Support TinyML.

Hardware	**Processor**	**CPU Clock**	**Flash**	**SRAM**
Apollo3	32-bit ARM Cortex-M4F	48 MHz, 96 MHz with TurboSPOTTM	1 MB	384 K
STM32F Discovery	32-bit ARM Cortex-M4 FPU Core	48 MHz	1 MB	192 K
ST IoTDiscovery	ARM Cortex-M4	48 MHz	1 MB,64Mbit Quad-SPI	128 K

Hardware	Processor	CPU Clock	Flash	SRAM
ECM3532 AI Sensor NSP	ARM Cortex-M3, NXP CoolFlux 16-bit DSP	100 MHz	512 KB	256 K
Arduino Nano 33 BLE Sense	nRF52840	64 MHz	1 MB	256 K
OpenMV Cam H7 Plus	ARM Cortex-M7	480 MHz	2 MB (Internal)	1 MB 32 MB SDRA
Himax EW-I Plus	32-bit ARC EM9D DSP with FPU Core	400 MHz	2 MB	2 MB
Thunderboard Sense 2	EFR32™ Mighty Gecko Wireless SoC	38.4 MHz	1 KB	256 K
Sony’s Spresense	ARM Cortex-M4F 6 Core	156 MHz	8 MB	1.5 M

Hardware	Processor	CPU Clock	Flash	SRAM
TinyML Board	Syntiant® NDP101 NDP, 32-bit ARM Cortex-M0	48 MHz	256 KB	32 KB
Arduino Portenta H7	ARM Cortex-M7, ARM Cortex-M4 GPU	480 MHz, 240 MHz	16 MB	8 MB SDRAM
Raspberry Pi 4B	64-bit ARM Cortex-A72 quad core, Broadcom BCM2711	1.5 GHz	–	256 K
AI-deck 1.1	GAP8, ESP32	168 MHz	1 MB	192 K
Pico4ML BLE	Raspberry Pi RP2040 DSP dual core	133 MHz	4 MB	264 K

Hardware	Processor	CPU Clock	Flash	SRAM
MKR Video 4000	Intel® Cyclone ® 10CL016 FPGA, , 32-bit ARM Cortex M0	48–200 MHz	2 MB, 256 KB	32 KB 8 MB SDRAM
MKR Video 4000	Intel® Cyclone ® 10CL016 FPGA,, 32-bit ARM Cortex M0	48–200 MHz	2 MB, 256 KB	32 KB 8 MB SDRAM
Nicla Sense ME	ARM Cortex M4	64 MHz	512 KB	64 KB
CC1352P Launchpad	CC1352R Wireless MCU LaunchPad™	48 MHz	352 KB	8 KB

Hardware	Processor	CPU Clock	Flash	SRAM
Hardware	Processor	CPU Clock	Flash	SRAM
ESP-EYE	32-bit ESP32	240 MHz	4 MB	8 MB PSRAM
GAP8	RISC-V hardware convolution engine	250 MHz (FC), 175 MHz (C), 22.65GOPs	512 KB	80 KB SDRAM
GAP9	RISC-V, hardware convolution engine	400 MHz, 150.8GOPs	1.5 MB	128 K 2 MB Externa
Nordic Semi nRF52840 DK	ARM Cortex M4	64 MHz	192 KB	24 KB
Nordic Semi	ARM	64 MHz	1 MB	256 K

Hardware	Processor	CPU Clock	Flash	SRAM
Thingy:91	Cortex M33, nRF9160 SiP			
XCore.ai	Convolution and dense neural network FPU 16 core	3200MIPS, 1 M 512 FFTs/s	–	1 MB
FRDM-K64F	ARM Cortex M4	120Mhz	1 MB	256 K

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, WioTerminal, 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 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).

- Download: Download high-res image (121KB)
- Download: Download full-size image

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

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.

. Download: Download high-res image (310KB)

. Download: Download full-size image

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

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. Furthermore, we 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).

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
Training	Weights	Class Hypervectors
Testing	Run Pre-trained Classifier	Associate Query Hypervec
Model Complexity	High	Low
Accuracy	Very High	Acceptable

Computing Types	Classical Computing	HDC Computing
Feature Encoding	Easy	Difficult
Number of Features	Many	One

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 hypervector (Q) is later generated during the inference process based on a given input as per the process mentioned for the sample hypervectors. At 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.

. Download: Download high-res image (126KB)

. Download: Download full-size image

Fig. 8. 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 second 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.

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

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.

. Download: Download high-res image (472KB)

. Download: Download full-size image

Fig. 10. SwapNN architecture.

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 condensers 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.

. Download: Download high-res image (76KB)

. Download: Download full-size image

Fig. 11. Attention condenser with a condensation layer.

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 ImageNet50 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.