Received 24 August 2022, accepted 6 September 2022, date of publication 16 September 2022, date of current version 28 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3207200

RESEARCH ARTICLE

Unlocking Edge Intelligence Through Tiny Machine Learning (TinyML)

SYED ALI RAZA ZAIDI[1], (Member, IEEE), ALI M. HAYAJNEH[2], (Member, IEEE), MARYAM HAFEEZ[3], (Member, IEEE), AND Q. Z. AHMED[3], (Member, IEEE)
[1]School of Electronic and Electrical Engineering, University of Leeds, LS2 9JT Leeds, U.K.
[2]Department of Electrical Engineering, Faculty of Engineering, The Hashemite University, Zarqa 13133, Jordan
[3]School of Computing & Engineering, University of Huddersfield, HD1 3DH Huddersfield, U.K.

Corresponding author: Syed Ali Raza Zaidi (s.a.zaidi@leeds.ac.uk)

This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant UK EP/S016813/1 and Grant EP/N010523/1; in part by the Royal Academy of Engineering, Transforming Systems through Partnership TSP1040, under Grant UK 122040; and in part by the Royal Academy through the Distinguished International Associates under Grant DIA-2021-18.

ABSTRACT Machine Learning (ML) on the edge is key to enabling a new breed of IoT and autonomous system applications. The departure from the traditional cloud-centric architecture means that new deployments can be more power-efficient, provide better privacy and reduce latency for inference. At the core of this paradigm is TinyML, a framework allowing the execution of ML models on low-power embedded devices. TinyML allows importing pre-trained ML models on the edge for providing ML-as-a-Service (MLaaS) to IoT devices. This article presents a TinyMLaaS (TMLaaS) architecture for future IoT deployments. The TMLaaS architecture inherently presents several design trade-offs in terms of energy consumption, security, privacy, and latency. We also present how TMLaaS architecture can be implemented, deployed, and maintained for large-scale IoT deployment. The feasibility of implementation for the TMLaaS architecture has been demonstrated with the help of a case study.

INDEX TERMS Tiny machine learning, IoT, edge computing, 5G, LoRa, gesture recognition, deep learning, transfer learning, federated learning, implementation, MLOps, energy efficiency.

I. INTRODUCTION

Artificial intelligence (AI) is fundamentally a collection of digital technologies that derive **cognition** (learning, planning and orientation) through **perception** of the environment, therefore providing capability to make the right decisions at the right time. **Machine Learning** (ML) is an algorithmic tool-set to enable AI, in this context cognition is implemented as a learning process based on experience in relation to a certain task with an associated performance measure. The experience is often derived from the data collected from the perception of the environment.

A. MOTIVATION

The symbiosis of **AI** and **Internet-of-Things** (IoT) is natural. IoT provides a perception layer for many smart-city applications and ML augments sensing capabilities by deriving intelligence from data collected through this perception layer. A typical IoT device will have a micro-controller unit (MCU) potentially with an integrated low-power radio transceiver for wireless connectivity. Each MCU is furnished with a limited amount of memory and is typically designed to operate on a coin cell for several years. These low-cost IoT devices have a small footprint as often designed to be less obtrusive. In current IoT architecture typically data from the end-nodes/IoT devices is transmitted and aggregated at the gateways. The transmission of data is supported by a variety of connectivity technologies ranging from Unlicensed WiFi and Low-Power Wide-Area-Networking (LPWAN) technologies such as LoRa and Sigfox, as well as Licensed Cellular radio e.g. LTE for Machines (LTE-M) and Narrow Band (NB) IoT. The gateways are then connected to Cloud platforms via a broadband internet connection and allow storage, processing and inference on the

The associate editor coordinating the review of this manuscript and approving it for publication was Wai-Keung Fung.

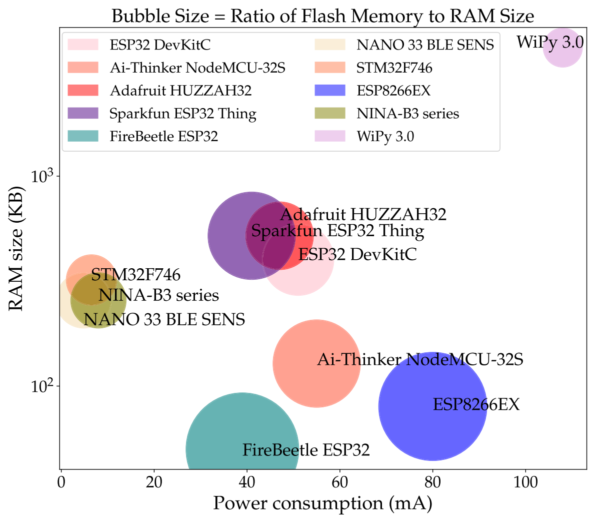

FIGURE 1. Power Consumption & Available Memory in Commercially available IoT MCUs.

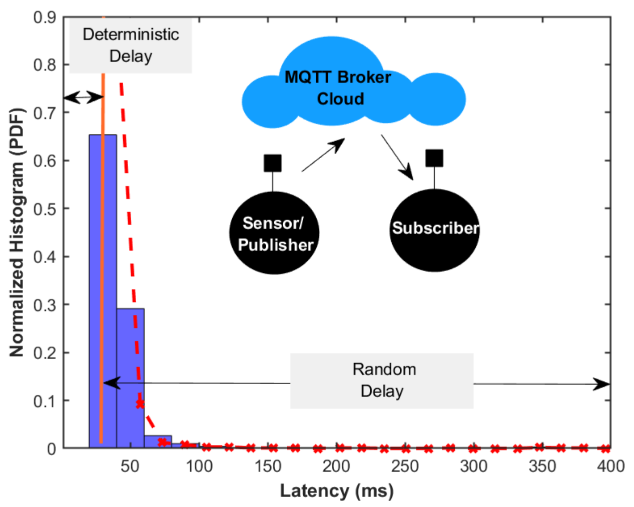

FIGURE 2. Latency for Typical MQTT IoT application.

aggregated sensor data. This architecture has several performance bottlenecks:

1) POWER CONSUMPTION FOR WIRELESS CONNECTIVITY

Most IoT devices are battery-powered allowing tetherless connectivity in scenarios where: i. fixed power supply does not exist; ii. the power source is not co-located with point-of-interests (PoI) for monitoring or actuation; iii. plugging into mains is costly and outweighs the device utility; and iv. mains power limits the mobility of the object in which IoT sensors are augmented. However, reliance on battery power means that operational lifetime is limited and power consumption is of paramount importance. Typically local computation is several-fold less power-hungry than transmission over wireless channel [1]. Moreover, while sensors collect a huge amount of data, from an application perspective it is the inference that is of importance. This inadvertently requires **compute efficient method of localized ML** which also increases bandwidth efficiency. Fig. 1 provides a quick comparison of popular commercially available IoT MCUs. Ideally, in order to provide local computation capabilities, MCUs with high onboard memory (RAM and FLASH) and low power consumption are required, i.e. the sweet spot for MCUs is the top-left corner of the figure. Unfortunately, it can be observed there are no commercial products that lie in that sweet spot. This could be credited to the inherent interplay between onboard memory and power consumption. Nevertheless, the industry has made rapid progress and some of the recent MCUs (STM, NINA and Nano BLE Sense) are placed in the next best space for design choices.

2) LATENCY

The current architecture not only incurs power cost to facilitate wireless connectivity but also introduces non-deterministic delay. Fig. 2 shows the probability density function (PDF) of latency for the Message Queuing Telemetry Transport (MQTT) packet published by an IoT device e.g. temperature sensor for a subscriber such as a remote monitoring tablet. Each packet is 140 bytes in size and latency is calculated at the application layer. It is obvious that there is a minimum fixed latency (which itself is a function of the load on the server, time of day, etc.) and a power-law distribution for the random delay. This scenario does not include the time taken to perform ML to derive inference. However, it shows that the round trip latency of cloud-centric solutions is unpredictable. Therefore, it is highly desirable to move the **inference capabilities** to the edge.

3) PRIVACY

Another disadvantage of cloud-centric architecture is that raw data from the sensor has to traverse to the cloud via gateway and the internet. Any compromise in security leads to privacy violations. Also, there is no transparency on how data is being used and which applications are authorized to use it. Moving inference closer to the user, i.e. on the edge can provision an architecture whereby raw data is never transmitted to the cloud. Only inferences driven on the raw data are transmitted to the cloud and stored for application-specific actions.

4) CONNECTIVITY

Lastly, in the current architecture gateways need to be connected to the internet. Provisioning such connectivity incurs both capital expenditure (CAPEX) and operational expenditure (OPEX) with management overheads associated with maintaining infrastructure. Local inference capabilities reduce the reliance on connectivity, i.e. provision of services in areas where internet connectivity is intermittent or even does not exist at all.

B. ML-AS-A-SERVICE (MLaaS)

The above-mentioned performance limitations have triggered a new paradigm for edge intelligence, i.e., MLaaS hosted on the edge for the IoT nodes. This emerging architecture is shown in Fig. 3. The architecture aims to employ on-device learning for low-fidelity ML, i.e. to perform rough and quick estimates. This is especially useful when actuation depends on such inference as ultra-low latency can be guaranteed. Additionally, the trained ML model which is deployed on the Edge-gateway can be exposed as a service for the end IoT devices. This provides the capability to perform high-fidelity ML at the edge. In many practical scenarios, the end devices or IoT devices will not perform any inference and will just access MLaaS from the edge. This allows for conserving energy on end nodes which are typically battery-powered, while gateways have more compute, memory and potential access to a fixed power supply. A step further from this architecture can be enabled through the exploitation of virtualization, i.e., the edge device can actually offload ML inference tasks horizontally between the end nodes. However, this increases gateway design complexity potentially affecting **continuous integration (CI)** and **continuous delivery (CD)** capabilities for firmware software. Moreover, the end nodes deplete more power. Nevertheless, in both aforementioned approaches, end-devices and gateways (typically embedded devices) need to be capable of implementing ML inference while coping with limited memory and power budget. **TinyML** precisely tries to address these design issues. In addition to ML capabilities, several other features need to be incorporated into the IoT Gateway under the proposed architecture. These include:

1) DEVICE MANAGER

Coordinates: a) commissioning and authentication for new IoT nodes added to the network; b) provides functionality to construct shallow or deep copies of device state, thereby enabling operation with intermittent connectivity. The commissioning functionality requires the implementation of either a lightweight web-based user interface (UI) that can be serviced by the gateway or a separate mobile application that can be downloaded by the user. Typically the approach taken in many commercial products is to host a lightweight web server on the gateway which can host the web-based UI. Additional UI functionality for visualization of sensor output and inference is often also incorporated in the web application.

2) CLOUD MANAGER

Coordinates: a) authentication of the gateway with the back end cloud service provider; b) provides cloud connectivity manager, e.g. management of connectivity to LPWAN; and, c) can additionally host a light-weight web-server which can allow pre-fetching of server-side rendered components for IoT applications, as well as rendering of data-visualization which can be locally performed.

3) RESOURCE MANAGER

Performs a) local storage management for core application logic, as well as visualization for the end-user device; b) performs memory management for both ML functions and end-user visualization.

The goal of this article is to provide a comprehensive overview of how TinyML-as-a-Service **(TMLaaS)** architecture and also demonstrate its usefulness through a practical case study. We also outline practical considerations in implementing the proposed architecture.

II. ML ON MCUs: CHALLENGES & SOLUTIONS

Neural network (NN) based learning has been extensively studied from an embedded application development perspective in [2], [3], [4], [5], [6], and [7]. Convolutional NN (CNN), Recurrent NN and their hybrids exploit the correlation of time-domain, spectral-domain features and long-term dependencies in the input data to improve the performance of the ML model. However, due to the high computational and storage requirement of the neural architecture of these models, full-scale deployment of many models is infeasible for IoT applications employing MCUs. In this context, **model compression** techniques are an active area of research to meet the hardware constraints. In [8], an RNN compression approach using pruning and integer quantization of weights/activation has been suggested for deployment on MCUs. Depthwise-separable (DS-CNNs) decompose 3-D convolution operations into 2-D and 1-D convolutions and have been suggested in [9] as an optimal choice of ML on MCUs. Another such effort is presented by [10] which is geared towards reducing the complexity of computation of convolutions in NNs by approximating the matrix multiplication calculations. However, it does introduce significant costs in terms of employing more additional operations. In addition to model compression, various optimizations based on adaptive NN architectures have also been proposed for provisioning ML models on MCUs. In [11], a two-stage neural architecture search approach has been proposed that first optimizes the search space to fit the resource constraints, and then specializes in the network architecture in the optimized search space. This allows the model to adapt to different QoS and resource constraints. Tree-based learning approaches [12] have been studied in the context of reducing computational load and storage requirements. Such models offer acceptable accuracy, however, they are limited in terms of their scalability. Hybrid approaches such as [13] combine DS- CNN neural architectures and tree-based learning to offer solutions that attain accuracy by using neural networks for feature extraction and computational efficiency by using a shallow Bonsai decision tree to perform the classification. TinyML framework can exploit neural architecture search (NAS) to design models that can meet the stringent MCU memory, latency, and energy constraints. An example of such an approach can be found in [14]. Articles [15] and [16] provide a comprehensive overview of the revolution of TinyML and a review of tinyML studies. In [17] authors present a new data compression solution called the Tiny Anomaly Compressor (TAC) to leverage the TinyML perspective. In [18], the authors present an architecture for detecting medical face masks

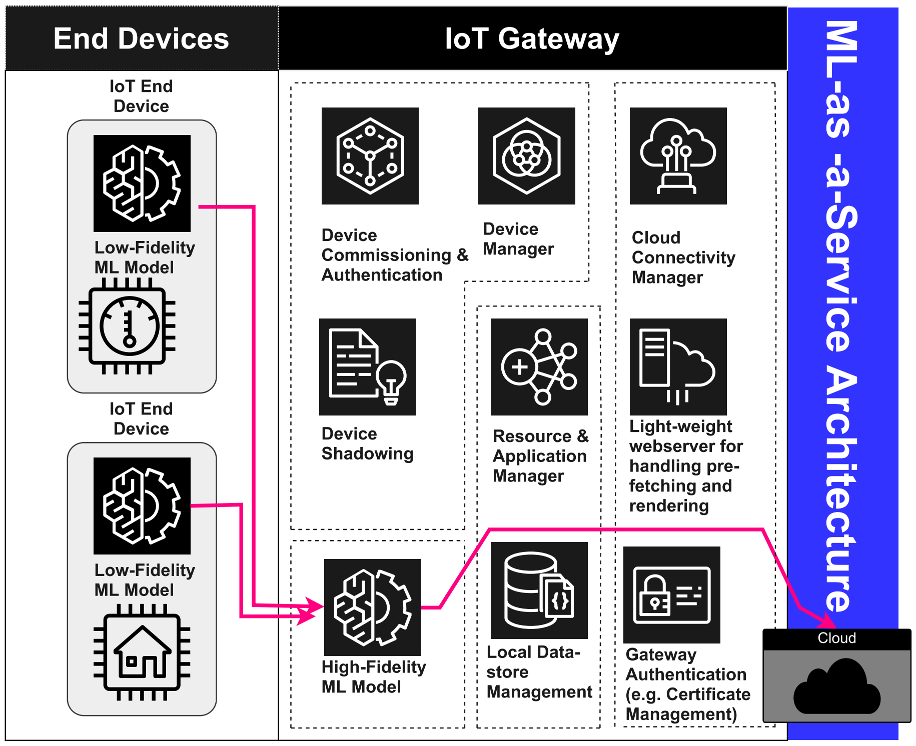
End Devices
IoT Gateway
ML-as -a-Service Architecture
IoT End Device
Low-Fidelity ML Model
IoT End Device
Low-Fidelity ML Model
Device Commissioning & Authentication
Device Manager
Cloud Connectivity Manager
Device Shadowing
Resource & Application Manager
Light-weight webserver for handling pre-fetching and rendering
High-Fidelity ML Model
Local Data-store Management
Gateway Authentication (e.g. Certificate Management)
Cloud

FIGURE 3. Architecture for the gateway.

for deployment on resource-constrained endpoints having extremely low memory footprints. However, to the best of our knowledge, multiple design trade-offs and implementation-related challenges involved in TMLaaS architecture need further attention as presented in the discussion to follow.

A. CONTRIBUTIONS

The main contributions of this work include a comprehensive framework for exploiting TinyML-as-a-Service (TMLaaS) at the Edge for future IoT applications. The paper identifies the need to have light ML solutions to fit the constrained onboard power and memory resources by providing an overview of onboard memory and power consumption of a variety of commercially available MCUs. The proposed TMLaaS architecture in this paper supports the on-device learning for low-fidelity ML as required for resource-constrained MCU-based IoT-type solutions. We present multiple design tradeoffs and implementation-related challenges involved in TMLaaS architecture. TFLite Micro-based design and workflow implementation of TinyML solutions is detailed and the operational cycle of the architecture has also been discussed. The paper also presents a practical case study based on the proposed design space of TMLaaS architecture. We outline some important research challenges and provide a brief road map for the evolution of TMLaaS-empowered IoT networks.

B. ORGANISATION

The rest of the paper is organized as follows: In Section II, we present a brief account of existing ML approaches and literature on implementing these models on resource-constrained MCUs. In Section III, the design and implementation challenges of interoperability and performance characterization to unlock the full potential of ML solutions has been discussed. TFLite Micro-based design and workflow implementation of TinyML solutions is detailed. Section IV explains MLaaS architecture in detail along with its operational cycle termed at ML-Ops. Section V presents the ease of development and deployment of the proposed TLMaaS architecture with the help of a simple case study. In Section VI, emerging ML modalities, road maps and open issues have been discussed with a focus on Transfer Learning and Federated Learning. Section VII concludes the paper while Section VIII details the future work.

III. TinyML OVERVIEW

TinyML is an emerging field at the intersection of embedded systems and ML. Effectively, TinyML provides tools to develop ML models which can be executed on resource-limited devices. The process flow for TinyML deployment starts with the collection of data from the hardware where an inference engine is required. The data can either be logged on onboard storage or could be directly imported into user-friendly tools e.g. Edge Impluse Studio. The ML model

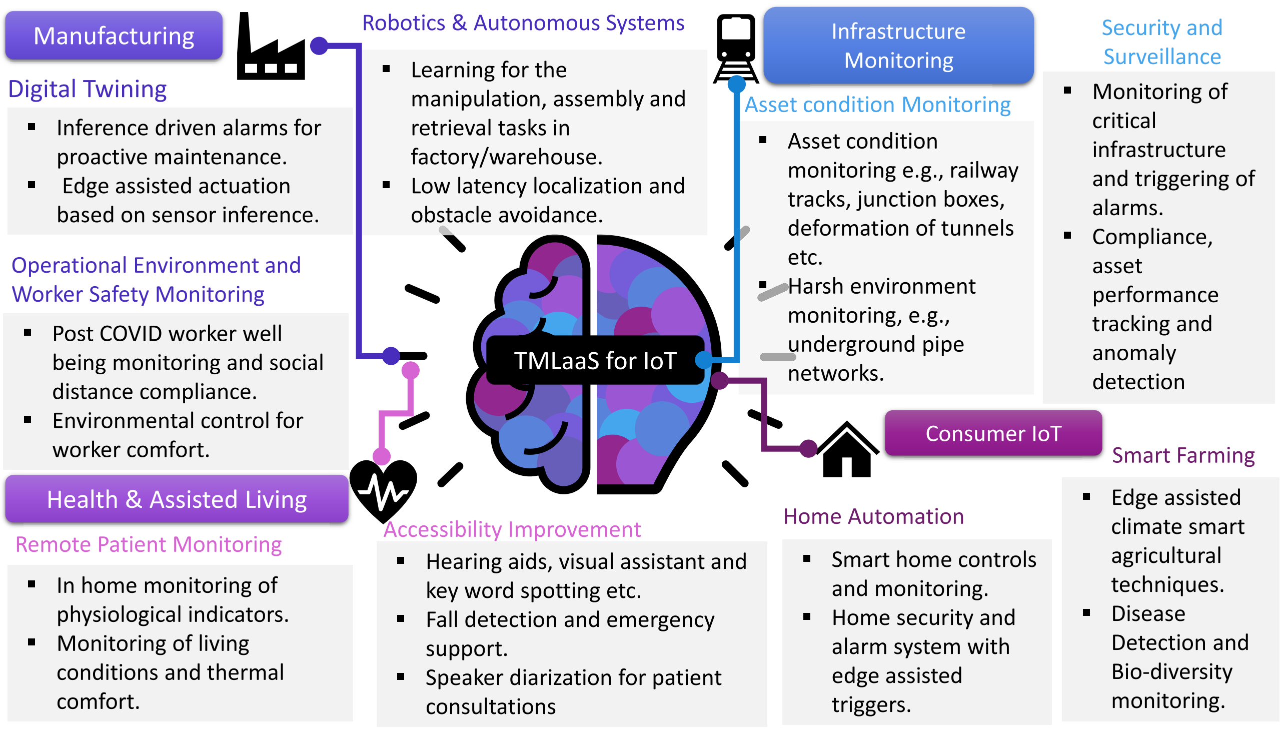
Manufacturing
Digital Twining
Inference driven alarms for proactive maintenance.
Edge assisted actuation based on sensor inference.
Operational Environment and Worker Safety Monitoring
Post COVID worker well being monitoring and social distance compliance.
Environmental control for worker comfort.
Health & Assisted Living
Remote Patient Monitoring
In home monitoring of physiological indicators.
Monitoring of living conditions and thermal comfort.
Robotics & Autonomous Systems
Learning for the manipulation, assembly and retrieval tasks in factory/warehouse.
Low latency localization and obstacle avoidance.
TMLaaS for IoT
Accessibility Improvement
Hearing aids, visual assistant and key word spotting etc.
Fall detection and emergency support.
Speaker diarization for patient consultations
Infrastructure Monitoring
Asset condition Monitoring
Asset condition monitoring e.g., railway tracks, junction boxes, deformation of tunnels etc.
Harsh environment monitoring, e.g., underground pipe networks.
Security and Surveillance
Monitoring of critical infrastructure and triggering of alarms.
Compliance, asset performance tracking and anomaly detection
Consumer IoT
Home Automation
Smart home controls and monitoring.
Home security and alarm system with edge assisted triggers.
Smart Farming
Edge assisted climate smart agricultural techniques.
Disease Detection and Bio-diversity monitoring.

FIGURE 4. Applications for TMLaaS.

is trained on the collected data set and the model is then exported into a form that is directly implementable on the MCU. The MCU employs the trained model for inference in subsequent iterations. In order to unlock the full potential of ML for IoT systems, two key challenges need to be addressed:

A. Interoperability: The MCU market is quite fragmented and there is no unified standard for TinyML implementation. Platform-specific implementations are not scalable as they require a manual application and hardware-specific optimizations.

B. Performance Characterization: Another challenge limiting TinyML is that in the absence of a uniform framework, it is hard to evaluate the hardware performance in a neutral and vendor-agnostic manner. When performance boosts are reported it is difficult to decipher whether they are related to hardware or software implementation and whether these gains generalize across various applications. These two challenges are addressed by a TensorFlow Lite (TFL) Micro proposed in [19]. In fact, TFLite has become synonymous to TinyML itself as most practical implementations of ML models rely on the TFLite libraries.

A. TinyML IMPLEMENTATION ASPECTS & DESIGN PRINCIPLES FOR TFLite MICRO

The core design principle behind TFLite Micro is that it aims to provide the utmost necessary features for enabling ML on IoT devices. In other words, the framework assumes that the model, input data and output arrays are in memory and perform computations on these arrays directly. The TFLite Micro Framework uses an interpreter which loads the data structure that defines the ML model. The choice of interpreter against the code generators is based on the fact that the model can be easily updated without the need for recompiling the firmware on the IoT device.

B. TinyML WORKFLOW IMPLEMENTATION

A typical TinyML workflow will include training of ML model using the TensorFlow framework on a high-performance computer (e.g. using TFLite within Jupyter Notebook) or on a cloud server (e.g. using Google Colab). The Tensor Flow model is then converted to a TFLite format using a converter. The converted model can then be exported as a C-byte array. On the MCU, the application utilizes the TinyML library which exposes the OpResolver application programming interface (API). In contrast to TensorFlow, TFLite Micro supports only a limited number of operations for implementing NNs. The application developer utilizes OpResolver to specify which operators need to be linked to the final binaries of the IoT firmware. This in turn minimizes the file size for the compiled firmware. Within the firmware, a contiguous chunk of memory "arena" needs to be allocated for TFLite Interpreter to store intermediate results and other variables. Typically, an instance of Interpreter is created by furnishing arena and OpResolver as a parameter. The Interpreter then takes care of memory allocation, operator resolution and model loading. The input data for the model is written onto an array which is read by the interpreter. Similarly, the output data is provided in form of an output array.

Interested readers are encouraged to see [20] for detailed tutorials on implementation.

IV. TMLaaS ARCHITECTURE

A. TinyML AS A SERVICE

TMLaaS refers to exposing the MLaaS on Gateways and End-Devices. Effectively, as shown in Fig. 3, the learning process can be viewed as a monolithic functionality split across the end device and the gateway. Devices perform **quick-and-coarse inference** while gateway can provide **fine grained** inference capabilities. TMLaas expands the ML capabilities provided by TinyML. In other words, TMLaaS is an architecture for exposing TinyML-as-a-Service. In contrast, TinyML alone provides a framework for enabling the execution of ML models on resource-constrained devices. Some of the key applications of TinyML have been presented in Fig. 4.

When designing the TMLaaS system an important question is how such a **split** can be accomplished. A simple answer to this design question is that for those applications where each IoT device contributes a **sub-set of input features** for the ML model, it is natural to expose the learning on the gateway. Also where these sensors are **heterogeneous** or the inference aims to treat the input from sensors as a multiple-input channel, it is easy to expose MLaaS on the gateway. On the other hand, in applications where inference leads to **actuation**, latency can be reduced by placing coarse inference capabilities on the device itself. Both choices yield different design **trade-offs**. For instance, a designer can think of trading onboard computations on devices with computation on the gateway. Both have different **energy penalties** with the later design option trades on-board energy consumption of computation with the energy consumed in wireless transmission. Depending on the computational requirements it might be better to run the model on the gateway which may be powered from mains rather than running these models on devices that are battery-powered. Another such trade-off is in terms of **security**. Depending on how secure the link between the device and gateway is versus the link between gateway and cloud, one might establish the split of the learning process. This of course needs to be considered along with **privacy constraints** enforced by the application. For instance, if the aggregate information at the gateway contains data from devices belonging to different users and certain devices provide side information for other features/data points then privacy aspects will dictate how coarse and fine-grained learning is split. Lastly, one needs to consider the **modality of communication** on devices and the ease of upgrading/replacing the ML model. Gateways might have a bi-directional communication link with the cloud while IoT devices can be uplink only (e.g. SigFox, ELTRES or BLE Advertising Beacons). Therefore, it is easier to upgrade and test the models on the gateway rather than on end devices. As long as the method with which the MLaaS is exposed, the end devices do not change the application will operate seamlessly. For instance, such functionality can be exposed using RESTful APIs. Maintaining, managing, testing and re-configuring the TMLaaS applications falls under MLOps. We present a brief overview of MLOps in what follows.

B. ML-OPS

In order to understand the operation of ML-based embedded applications, it is important to understand ML systems development life cycle and aspects of the continuous high-quality operation. This corresponds to a complete ecosystem of ML-embedded solutions from development through to delivery. This includes system construction, its integration, testing, releasing, deployment and infrastructure management [21]. Deriving from this definition, MLOps is a set of engineering practices that aim at unifying ML system development and operation with the objective of ensuring continuous integration (CI), continuous development (CD) and continuous testing (CT). In principle, MLOps is equivalent of DevOps i.e. combination of practices and tools that increases the ability of the software provider to deliver applications and services at a rapid pace. The following steps provide a footprint of activities that may be used as a reference classification for MLOps: (i) **Sampling:** For embedded applications, it is of utmost importance to have samples of data arriving at an accurate rate for processing. This is particularly important for real-time embedded applications using data from multiple sources; (ii) **Analysis:** It involves the classification of data based on the model schema and its requirements. It also includes the metadata which can be of higher value as compared to raw data hence a key enabler for federated learning; (iii) **Structure:** The data needs to be formatted in accordance with the ML model input structure. It needs to be partitioned into training and validation sets; (iv) **Training:** The TinyML model is trained using the structured data; (v) **Evaluation and Validation:** The model is evaluated and validated against test data and benchmarks; (vi) **Deployment:** The validated model is deployed; (vii) **CI:** It refers to multiple integrations on a daily basis. These can include algorithmic as well as visualization/firmware updates; (viii) **CD:** Based on the CI, development is also continuous with updates based on changing application dynamics or algorithms. New models can be developed and trained. Network bandwidths, latency and privacy are key aspects of TinyML performance that may require a CD approach to guarantee QoS; (ix) **CT:** Continuous validation and testing of the updated models.

V. CASE STUDY

In this section, we aim to demonstrate the ease of **development** and **deployment** of the proposed TLMaaS architecture with the help of a simple case study. The case study will showcase how some of the features previously outlined can be implemented in practice, as well as present a benchmark of different ML models on the hardware. Through this case study, we explore the feasibility of implementing the most commonly used classical classification and deep neural network (DNN) techniques.

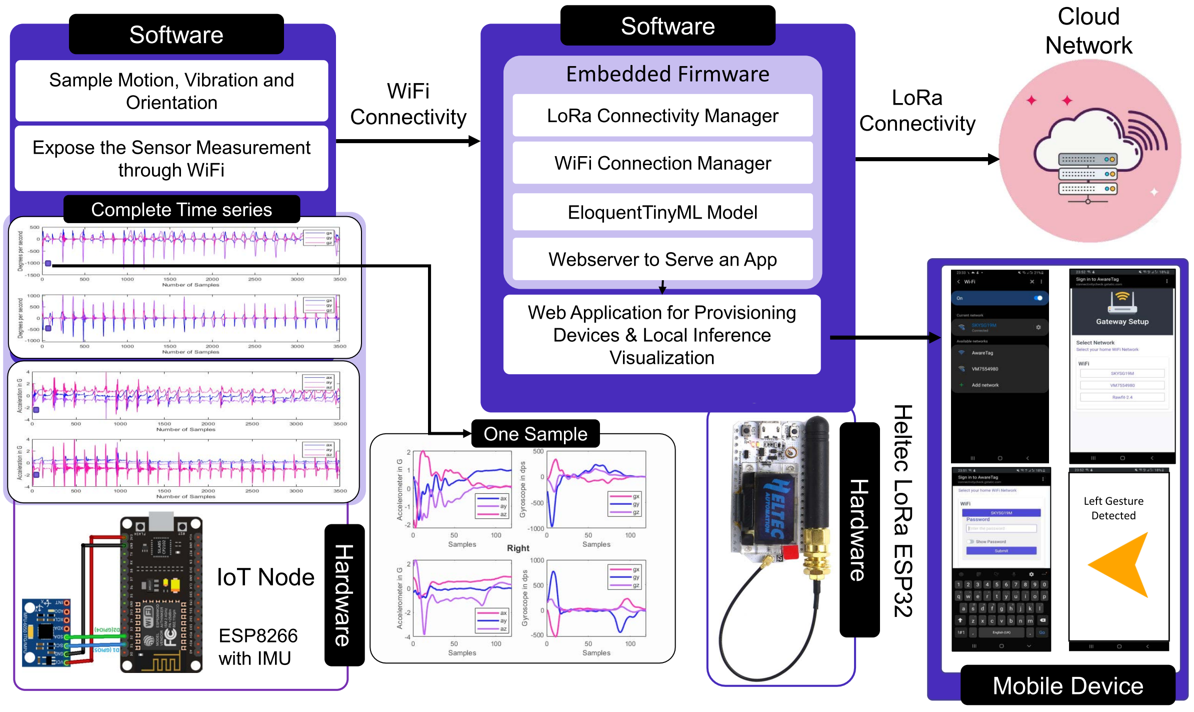

FIGURE 5. Case study setup.

A. SCENARIO

We consider a scenario where a wearable device is equipped with an inertial measurement unit (IMU) and WiFi transceiver. The gateway device receives the data sent by the wearable device. The wearable device has a sampling trigger, i.e., it starts constructing an observation window after a certain acceleration threshold is exceeded. The observation window has a fixed length which is considered a design variable. Once the observation window is fully constructed data is transmitted over WiFi to the gateway device. The gateway device receives raw data which is treated as input to the ML model. This data is used in conjunction with a pre-compiled model by TinyML interpreter to produce the classification output. In particular, we aim to classify two different gestures, i.e. left and right. The choice of these gestures is on the basis that we can construct a wearable joystick that can be used for a variety of applications (e.g. presentations, gaming etc.). Notice the choice of which gestures, we want to classify is arbitrary and mainly based on ease of collection of data. However, the same principle applies if, for instance, we want to use IMU data to detect falls in the elderly. To provide a comparison of this case study with a standard example, we employ the MNIST dataset for the classification of handwritten digits. Studies in [22] have used a single-layer CNN-based model for gesture recognition. In [23], Tensorflow (TF) and Keras are evaluated for their usage in gesture recognition. The gestures classification dataset samples consist of the two three-dimensional sensor measurements (i.e., accelerometer and gyroscope) that add up to six raw features. The target of the classifier is simply to label the class of the gesture whether it is left or right based on the selected features. For the MNIST dataset, the aim is to label the handwritten 28 × 28 pixels number into the actual value from 0 to 9.

B. IMPLEMENTATION

The implementation of the case study in terms of hardware and software components is shown in Fig. 5. We employ NodeMCU (ESP8266) as an IoT device connected with an I2C interface to the IMU chip (MPU-6050). ESP8266 is set up as a WiFi client device and streams the six raw features from the IMU to the gateway. The gateway firmware scans for the data but also provides a sub-set of functionalities outlined in Fig. 3. For the gateway, we employ HELTEC LoRa Wireless Stick. The gateway has dual connectivity, i.e. WiFi and LoRa. The gateway implements: i) **Device Manager:** which handles the data streamed by all devices. It also manages connectivity between devices and the gateway. ii) **Lightweight Web Server:** A lightweight web server that hosts a web application written using React a JavaScript framework. The application provides functionality to provision either WiFi or LoRa connectivity to the cloud. The web application also provides visualization for the outcome of the local inference process on a per-device basis. The web application interacts with the web server which is fundamentally a part of the firmware through RESTful API.iii) **TMLaaS Engine:** this is implemented in the firmware using EloquentTinyML wrapper on TFLite. This allows the execution of TFLite on ESP32 MCUs present on the HELTEC board. The implementation process for TMLaaS functionality is comprised of three steps. First, we collect the training data to train the classifiers offline while

TABLE 1. DNN results.

Dataset	Input layer	Hidden layer	Output	RAM (kb)	Flash (kb)	Prediction (ms)	Accuracy
Gesture detection DNN	714x20	10x1	1	42.339	147.839	20.16	99%
	714x18	9x1	1	40.324	145.824	18.96	99%
	714x16	8x1	1	38.324	143.824	16.032	99%
	714x14	7x1	1	36.339	141.839	14.062	99%
	714x12	6x1	1	34.371	139.871	11.972	99%
	714x10	5x1	1	32.417	137.917	10.0133	99%
	714x8	4x1	1	30.484	135.984	8.0133	99%
	714x6	3x1	1	28.558	134.058	6.045	99%
MNIST DNN	64x56	56X16	10	40.027	145.777	8.577	95.72%
	64x48	48X16	10	37.496	142.964	7.635	97.01%
	64x40	40X16	10	34.964	140.433	6.555	98.64%
	64x32	32X16	10	32.343	137.812	5.205	97.22%

optimizing the hyper-parameters of the model and ensuring targeted accuracy is obtained. Secondly, we use the conversion techniques (i.e., TF Lite and EloquentML [24] Wrapper) that ports the model from the training programming language into a compiled byte array that can be used by TinyML Interpreter. This step includes ensuring that the compiled model will fit on the small flash memory footprint and can execute on the available SRAM. If the compiled model size is larger than any of the available resources, we then will need to sacrifice some of the optimized parameters and as a result the accuracy of the classifier. Third, the compiled model is ported to the MCU for live testing on unseen data in real time.

C. RESULTS & DESIGN INSIGHTS

Table 1 summarizes the performance for both data sets with different combination of hyper-parameters. The choice of hyper-parameters impact latency for prediction and the amount of RAM and Flash memory occupied by the model. However, the prediction accuracy has only a slight change if any with a reduction in the size of the input layer. Effectively, in this case, study (as can be seen from Fig 5) the raw waveforms have a distinct signature for both gestures. Therefore it is easy to obtain higher classification accuracy without a massive input layer and a huge number of hidden layers. A similar observation is valid for the MNIST data set. In summary, with the help of such analysis, we can establish minimum hyperparameter space which yields good accuracy while minimizing the prediction delay, memory and processing footprint on the gateway. Table 1 provides a summary of performance for different classifiers when employed on the two datasets of interest. Both Support Vector Machine (SVM) and Random Forest Classifiers provide good accuracy for the classification task. Nevertheless, the prediction time for Random Forest is several folds lower than SVMs. Random Forest Classifier also has a low resource footprint in terms of RAM and Flash space. Therefore, it seems like an ideal choice for implementing the classification task on the gateway using the traditional classification approach.

VI. EMERGING ML MODALITIES, ROAD MAP AND OPEN CHALLENGES

Although the proposed TMLaaS architecture itself is in its infancy, there are two key ML modalities and relevant open challenges that need to be addressed by the community. In our subsequent discussion, we outline what these ML modalities are and highlight their importance for enabling the pervasive deployment of TMLaaS architecture.

A. TinyML FOR TRANSFER LEARNING

A typical TinyML workflow involves training a model using high-performance computing platforms and then compiling it through the TinyML framework which can be interpreted on the edge device for making an inference. As an example, the model developed by OpenAI recently to solve rubrics cube not only required 1K desktop computers with several graphical processing unit (GPU) accelerators but also consumed 2.8 Gigawatt-hours of electricity. The evolving trajectory to expedite TMLaaS at the edge is to supply pre-trained models through the edge device and can be employed for inference. repository (e.g. TensorFlow Hub). These pre-trained models can then be **imported** by edge devices for inference. It is not always possible to train and re-train to build TinyML models because (a) Training on large datasets is computationally expensive; (b) The input data for certain use cases are not available but the trained models are often available. Additionally, **Transfer Learning** (TL) provides the capability of applying knowledge gained while solving one problem and applying it to different but semantically similar problems. Therefore, a pre-trained model on a semantically similar task can be easily downloaded from the hub/repository for the edge device and can then be employed for inference. TL scenarios may be **Inductive** (i.e., source and target problem domains are same but tasks are different), **Unsupervised** (i.e. lack of labeled data in target domain), or **Transductive** (semantic similarities in source and target problems but different domains for inputs). Since TMLaaS architecture is still in its infancy, the adoption of on-the-fly precompiled ML models through TL remains an open research area.

TABLE 2. Classifiers results.

Dataset	Classifier type	RAM (kb)	Flash (kB)	Prediction time (ms)	Accuracy
Gestures	SVM	2.898	45.26	13.474	94.70%
	Logistic Regression	2.898	13.88	0.698	94.70%
	GaussianNB	2.902	20.17	27.556	98.00%
	Decision Tree Classifier	2.898	9.10	0.0001	89.47%
	Random Forest Classifier	2.898	14.77	0.085	98.00%
MNIST	SVM	0.609	173.08	80.794	95.00%
	Logistic Regression	0.359	18.94	1.657	91.00%
	Gaussian NB	0.363	18.83	36.750	81.00%
	Decision Tree Classifier	0.1093	5.96	0.003	78.00%
	Random Forest Classifier	0.1093	5.96	0.030	91.50%

B. TinyML FOR FEDERATED LEARNING

As discussed earlier, TMLaaS architecture inherently allows the designer to preserve privacy by utilizing a pre-compiled model on the edge devices. However, the model itself does not improve over time, i.e. learning from other edge devices is not inherently feasible. In contrast, **Federated Learning** (FL) relies on distributed training model whereby data never leaves the edge devices. However, a shared model learns by aggregating locally computed ML models. In [25] authors introduce the Federated Averaging Algorithm, which combines local stochastic gradient descent (SGD) on each client with the server which then utilizes local computation for averaging. Although the initial model was proposed for identically and independently distributed (iid) datasets, recent works have extended the algorithm for the non-iid datasets. The combination of TinyML and Federated learning is natural as both complement each other perfectly. While TinyML can allow local execution of the models, FL can provide means for exploiting cross-device knowledge to improve the prediction model itself. To the best of our knowledge, the only attempt made to combine these two is in [26] and this is still an open area for future research.

VII. CONCLUSION

In this article, we presented a comprehensive framework for exploiting TinyML-as-a-Service (TMLaaS) at the Edge for future IoT applications. The paper compares onboard memory and power consumption of a variety of commercially available MCUs and identifies the need to have light ML solutions to fit their constrained power and memory resources. The outlined TMLaaS architecture supports on-device learning for low-fidelity ML. We presented several design tradeoffs and implementation-related challenges involved in TMLaaS architecture. TFLite Micro-based design and workflow implementation of TinyML solutions is detailed and the operational cycle of the architecture has also been discussed. The design space of TMLaaS architecture was also explored via a practical case study. Lastly, we outlined some important research challenges and a brief road map for the evolution of TMLaaS-empowered IoT networks.

VIII. FUTURE WORK

As a part of the future work in this area, several challenges need to be addressed for both TL and FL-based approaches before they can play their significant role in unleashing wide-scale penetration of TMLaaS solutions. The three key research challenges in the TL area are: a. How can the edge device evaluate the trustworthiness of a pre-trained model, i.e., how can an edge device be sure that the source domain data set was representative and did not have any biases? b. How can we build a trustworthy dissemination protocol for sharing retrained TL models between edge devices? c. The third and the most important design question is how we develop an interpretable TL model, i.e. transition towards. Explainable ML models for TMLaaS architecture. From FL perspective, some of the important design questions include: a. TinyML minimizes the energy cost paid in the transmission of data to the cloud by allowing edge execution of the ML model. However, for federated averaging at least the local updates of the model need to be transmitted to the Cloud. While this does not compromise privacy, it has energy and connectivity costs. A typical FL update workflow requires bandwidth that far exceeds the capability of current LPWAN technologies. Therefore, it is these trade-offs that will dictate when and how these two frameworks should be utilized in conjunctions. b. It is possible to compress FL models using ML compression techniques. TinyML frameworks e.g. TensorFlow Lite needs to explore how communication efficient model compression can be accomplished for FL.

In summary, both TL and FL will play an instrumental role in the proliferation of TMLaaS architecture. The architecture itself when viewed through the lens of these ML modalities provides a rich design space for further research. We hope that the highlighted design issues will trigger community interest in exploring some of these open research areas further.

REFERENCES

[1] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4403–4417, May 2020.

[2] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2014, pp. 4087–4091.

[3] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," Google Res., New York, NY, USA, Tech. Rep. 43969, 2015. [Online]. Available: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43969.pdf

[4] S. O. Arık, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional recurrent neural networks for small-footprint keyword spotting," in *Proc. Interspeech*, 2017, pp. 1606–1610.

[5] M. Sun, A. Raju, G. Tucker, S. Panchapagesan, G. Fu, A. Mandal, S. Matsoukas, N. Strom, and S. Vitaladevuni, "Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting," in *Proc. IEEE Spoken Lang. Technol. Workshop (SLT)*, San Diego, CA, USA, Dec. 2016, pp. 474–480.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[7] J. G. Wilpon, L. Rabiner, C.-C. Lee, and E. R. Goldman, "Automatic recognition of keywords in unconstrained speech using hidden Markov models," *IEEE Trans. Acoust., Speech Signal Process.*, vol. 38, no. 11, pp. 1870–1878, Nov. 1990.

[8] I. Fedorov, M. Stamenovic, C. Jensen, L.-C. Yang, A. Mandell, Y. Gan, M. Mattina, and P. N. Whatmough, "TinyLSTMs: Efficient neural speech enhancement for hearing aids," 2020, *arXiv:2005.11138*.

[9] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," 2017, *arXiv:1711.07128*.

[10] M. Tschannen, A. Khanna, and A. Anandkumar, "StrassenNets: Deep learning with a multiplication budget," in *Proc. 35th Int. Conf. Mach. Learn.*, Stockholm, Sweden, 2018, pp. 4985–4994.

[11] J. Lin, W.-M. Chen, Y. Lin, C. Gan, and S. Han, "McuNet: Tiny deep learning on IoT devices," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 11711–11722.

[12] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2 KB RAM for the Internet of Things," in *Proc. 34th Int. Conf. Mach. Learn.*, Sydney, NSW, Australia, 2017, pp. 1935–1944.

[13] D. Gope, G. Dasika, and M. Mattina, "Ternary hybrid neural-tree networks for highly constrained IoT applications," in *Proc. Mach. Learn. Syst. (SysML) Conf.*, Palo Alto, CA, USA, vol. 1, 2019, pp. 190–200.

[14] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. Whatmough, "MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers," in *Proc. Mach. Learn. Syst.*, San Jose, CA, USA, vol. 3, 2021, pp. 517–532.

[15] N. N. Alajlan and D. M. Ibrahim, "TinyML: Enabling of inference deep learning models on ultra-low-power IoT edge devices for AI applications," *Micromachines*, vol. 13, no. 6, p. 851, 2022.

[16] L. Dutta and S. Bharali, "TinyML meets IoT: A comprehensive survey," *Internet Things*, vol. 16, Dec. 2021, Art. no. 100461.

[17] G. Signoretti, M. Silva, P. Andrade, I. Silva, E. Sisinni, and P. Ferrari, "An evolving TinyML compression algorithm for IoT environments based on data eccentricity," *Sensors*, vol. 21, no. 12, p. 4153, 2021.

[18] P. Mohan, A. J. Paul, and A. Chirania, "A tiny CNN architecture for medical face mask detection for resource-constrained endpoints," in *Innovations in Electrical and Electronic Engineering*. Singapore: Springer, 2021, pp. 657–670.

[19] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, P. Warden, and R. Rhodes, "TensorFlow lite micro: Embedded machine learning for TinyML systems," in *Proc. Mach. Learn. Syst.*, vol. 3, pp. 800–811, 2021.

[20] T. F. Foundation. *Tflite Micro Implementation Tutorials*. Accessed: Aug. 23, 2022. [Online]. Available: https://bit.ly/2PGReuc

[21] G. C. Foundation. *MLOPS: Continuous Delivery and Automation Pipelines in Machine Learning*. Accessed: Aug. 23, 2022. [Online]. Available: https://bit.ly/2QH21VM

[22] S. Bian and P. Lukowicz, "Capacitive sensing based on-board hand gesture recognition with TinyML," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2021, pp. 4–5.

[23] B. Coffen and M. S. Mahmud, "TinyDL: Edge computing and deep learning based real-time hand gesture recognition using wearable sensor," in *Proc. IEEE Int. Conf. E-Health Netw., Appl. Services (HEALTHCOM)*, Shenzhen, China, Mar. 2021, pp. 1–6.

[24] Eloquent. *Eloquent TinyML Wrapper Library*. Accessed: Aug. 23, 2022. [Online]. Available: https://bit.ly/2O5FQYB

[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.* Lauderdale, FL, USA, 2017, pp. 1273–1282.

[26] K. Kopparapu and E. Lin. *TinyFL: Enabling Federated Learning on Tiny Devices*. Accessed: Aug. 23, 2022. [Online]. Available: https://bit.ly/3sMXa3P

SYED ALI RAZA ZAIDI (Member, IEEE) received the Ph.D. degree from the School of Electronic and Electrical Engineering. From 2011 to 2013, he was associated with the International University of Rabat working as a Research Associate. He was also a Visiting Research Scientist at the Qatar Innovations and Mobility Centre, from October 2013 to December 2013, working on QNRF Funded Project QSON. From 2013 to 2015, he was associated with the SPCOM Research Group working on US ARL Funded Project in the area of network science. He is currently an Associate Professor at the University of Leeds in the broad area of communication and sensing for robotics and autonomous systems. He has published more than 90 papers in leading IEEE conferences and journals. His current research interests include intersection ICT, applied mathematics, mobile computing, and embedded systems implementation. Specifically, his current research is geared towards: design and implementation of communication protocols to enable various applications (rehabilitation, healthcare, manufacturing, and surveillance) of future RAS; and design, implementation, and control of RAS for enabling future wireless networks (autonomous deployment, management, and repair of future cellular networks). He was awarded the G. W. and F. W. Carter Prize for best thesis and best research paper. He has been awarded COST IC0902, Royal Academy of Engineering, EPSRC, Horizon EU, and DAAD grants to promote his research outputs. From 2014 to 2015, he was an Editor of IEEE COMMUNICATIONS LETTERS and also a lead Guest Editor of *IET Signal Processing* journal's Special Issue on Signal Processing for Large Scale 5G Wireless Networks. He is also an Editor of *IET Access, Front haul and Backhaul* books. He is currently serving as an Associate Technical Editor for *IEEE Communication Magazine*.

ALI M. HAYAJNEH (Member, IEEE) received the B.Sc. and M.Sc. degrees from the Jordan University of Science and Technology (JUST), Irbid, Jordan, in 2010 and 2014, respectively, and the Ph.D. degree from the University of Leeds, Leeds, U.K. He is currently with the Department of Electrical Engineering, Faculty of Engineering, The Hashemite University, Zarqa, Jordan, where he is also the Director of the Innovation and Entrepreneurial Projects Center. He is working as an Android Development Freelancer, by integrating, and utilizing the IoT promising technologies for the management of smart homes and cities. His experience in many programming languages gives him the motivation to diversify his interests in utilizing many platforms and technology engines. His current research is funded by the Royal Academy of Engineering through two programs: Transfer Systems through Partnerships (TSP); and distinguished international associate (DIA) in the fields of smart agriculture, drone-assisted micro irrigation, and tiny machine learning on the edge IoT devices. His current research interests include drone assisted wireless communications, public safety communication networks, backscatter communication, deep learning, power harvesting, stochastic geometry, device to device (D2D), machine to machine (M2M) communications, modeling of heterogeneous networks, cognitive radio networks, and cooperative relay networks.

MARYAM HAFEEZ (Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Leeds, U.K., in 2015. From 2015 to 2018, she was a Research Fellow at the Institute of Robotics, Autonomous Systems and Sensing (IRASS), University of Leeds. Since 2018, she has been a Senior Lecturer at the Department of Engineering and Technology, University of Huddersfield. Her current research is funded by the EU Horizon 2020 program. Her research interests include the design and analysis of protocols for next-generation green intelligent wireless networks by employing tools from game theory and stochastic geometry along with the Internet of Things (IoT) and Industry-4.0-related research. She worked in the area of dynamic spectrum access had received the Best Paper Award from the IEEE International Conference on Communications (ICC), in 2013. She is also serving as a member of the Editorial Board for *Frontiers in Communications and Networks* journal.

Q. Z. AHMED (Member, IEEE) received the degree in electrical engineering from the National University of Science and Technology, Pakistan, in 2001, the M.Sc. degree in electrical engineering from the University of Southern California (USC), Los Angeles, USA, in 2005, and the Ph.D. degree in electronics and electrical engineering from the University of Southampton, U.K., in 2009. From 2009 to 2011, he was working as an Assistant Professor at the National University of Computer and Emerging Sciences (NUCES), Islamabad, Pakistan. In 2011, he started working as a Postdoctoral Fellow in cooperative communications at the King Abdullah University of Science and Technology (KAUST). In 2015, he started working as a Lecturer at the University of Kent, where he was involved with research on millimetre wave and device to device communications. He is currently a Reader in electronic and electrical engineering at the University of Huddersfield, U.K. He contributed to project management and research in EU FP7 and Horizon 2020 research projects, iCIRRUS and RAPID. He was the Main Investigator on Royal Society and Royal Engineering Project as well. In 2017, he joined the University of Huddersfield. He is presently the main author and co-investigator in EU H2020 ETN Research Project ''MObility and Training fOR beyond 5G Ecosystems (MOTOR5G)'' (3.9M€) and EU H2020 RISE Research Project ''Research Collaboration and Mobility for Beyond 5G Future Wireless Networks (RECOMBINE)'' (0.5M€). In August 2020, ''Capacity building for Digital Health monitoring and Care System'' is granted by the Erasmus+: Higher Education-International Capacity Building. Out of 1005 eligible applications submitted 164 have been selected for funding. The funding amount is approximatly 1M€. In June 2021, ''Smart Mattress'' has been granted 175,734 in collaboration with Deluxe Beds LTD by the Innovative UK. The project proposes to increase the mattress life cycle, which will reduce the number of mattresses sent to the landfill and improve our sleep quality.