# Poster Abstract: Approach for Remote, On-Demand Loading and Execution of TensorFlow Lite ML Models on Arduino IoT Boards

Bharath Sudharsan
Data Science Institute,
NUI Galway, Ireland
b.sudharsan1@nuigalway.ie

Simone Salerno
Eloquent Arduino,
Bari, Italy
eloquentarduino@gmail.com

Piyush Yadav
Collins Aerospace ART,
Cork, Ireland
piyush.yadav@collins.com

John G. Breslin
Data Science Institute,
NUI Galway, Ireland
john.breslin@nuigalway.ie

## ABSTRACT

Traditionally, original equipment manufacturers (OEMs) send device-specific over-the-air (OTA) packages to ensure the latest firmware, security patches, etc. With millions of IoT devices, even a tiny percentage of OTA failures will result in tens of thousands of globally affected consumers. The state-of-the-art OTA methods are suited for high-end Android & embedded Linux devices and not for resource-constrained devices (e.g. MCUs, small CPUs) with only a few MB memory. Current OTA methods have been tested only on non-ML use-cases such as remote bugs patching or security flaws, etc.

In this paper, we present OTA-TinyML approach that, via HTTPS, loads the C source file of ML models from a webserver into IoT boards. OTA-TinyML does not strain hardware (resource-friendly) as its implementation spans only a few lines of code. It is compatible with a range of ML models (e.g. text, speech, image domains) and MCUs (e.g. Cortex M series, STM32, Xtensa). OTA-TinyML is tested by performing remote fetching of 6 types of ML models, storing them on 4 types of memory units, then loading and executing on 7 popular Arduino IoT boards.

## KEYWORDS

IoT Devices, TinyML, OTA Updates, Device Repurposing.

## 1 INTRODUCTION

Edge-to-cloud OTA programming or updates offer the potential for enhanced revenue streams, as the OEMs can offer add-on services without having expensive service technicians or inexperienced users perform the updates in person [1]. However, these benefits must be balanced by the risks: a poorly executed OTA update can result in bricked devices and significant inconvenience to consumers, as well as reputational damage to the OEM [2]. Also, the majority of IoT devices are embedded systems with a low-cost MCU or a small CPU as their brain (market estimates 50 billion tiny chipsets shipped in 2020) [3, 4]. The hardware of such devices has processor chipsets from various vendors (heterogeneity introduces complexity), limited/no physical access to re-programming (concealed peripherals, ports, test points), and memory-constraint (few MB of SRAM, flash sufficient only for device routine functionalities) [5]. Such challenges impact the OTA update/programming process, increasing cases reporting that tiny devices get into an inconsistent state [6]. The state-of-the-art OTA methods with update version rollback, integrity verification, update failure management features are not suitable for MCU devices due to their memory and computation demands. OTA ML for tiny IoT devices research challenges are covered in [7]. We present OTA-TinyML, a novel approach that enables even constrained, low-cost IoT devices to perform end-to-end remote fetching, storing, and execution of ML models.

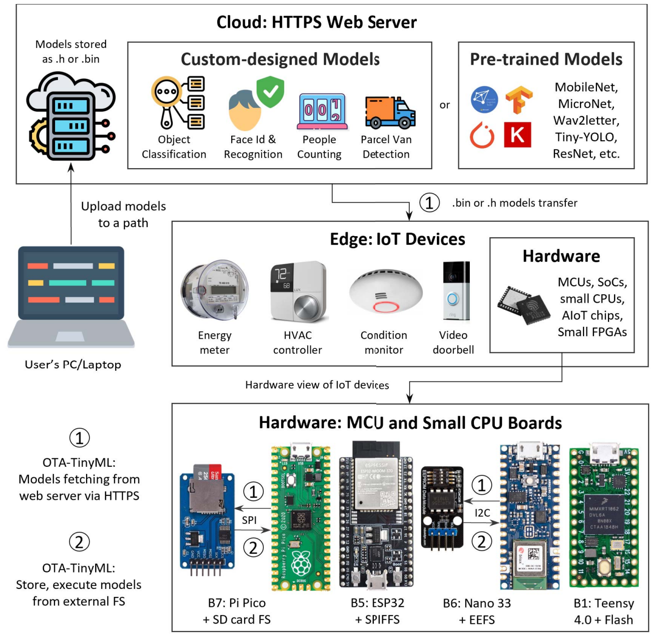

**Figure 1: OTA-TinyML for model fetching from a webserver, storing in external FS, and execution on IoT devices.**

## 2 OTA-TINYML DESIGN

The operational flow of OTA-TinyML is shown in Figure 1, which comprises of two parts. The first part circled ①, contains a method, that upon demand, fetches ML model files from the cloud server on the edge devices. The second part circled ②, contains a method to enable storage of fetched files in internal memory or external filesystems, then the loading and model execution. The next subsection discusses these parts in detail.
**Code**: https://github.com/bharathsudharsan/OTA-TinyML

### 2.1 Part 1: Models Fetching via HTTPS

This part of OTA-TinyML enables IoT devices to download ML models from the internet, whose implementation, via target server address along with the directory/path of ML model, initially establishes a connection to the server. Then downloads the target from HTTPS URL using `http.get()` method of HTTPClient object and passes the file to OTA-TinyML part 2 for storage on the available memory unit of the edge device. The models stored in the webserver need to be of the `.bin` (model as a compressed binary file) or `.h` format (model as a C array), both of which can be generated from the trained model using Converter API [8].

978-1-6654-9624-7/22/$31.00 ©2022 IEEE
DOI 10.1109/IPSN54338.2022.00061

**Table 1: Popular MCU boards (left) and optimized INT8 models (right) used for OTA-TinyML testing.**

| Name | Processor, Flash (MB), SRAM, Clock (MHz) |
|---|---|
| B1: Teensy 4.0 | Cortex-M7, 2, 1MB, 600 |
| B2: STM32 Nucleo H7 | Cortex-M7, 2, 1 MB, 480 |
| B3: Arduino Portenta | Cortex-M7+M4, 16, 8MB, 480 |
| B4: Feather M4 Express | Cortex-M4, 2, 192KB, 120 |
| B5: Generic ESP32 | Xtensa LX6, 4, 520KB, 240 |
| B6: Arduino Nano 33 | Cortex-M4, 1, 256KB, 64 |
| B7: Raspberry Pi Pico | Cortex-M0+, 16, 264KB, 133 |

| Task: Model Name | Score | .tflite (KB) | .h (KB) |
|---|---|---|---|
| Recognize Gestures: MagicWand | 0.67 (Acc) | 19 | 118 |
| Visual Wake Words: MicroNet S-L | 0.76-0.82 (Acc) | 273-529 | 1689-3267 |
| Speech Recognition: Wav2letter | 0.0783 (LER) | 22600 | 143421 |
| Keyword Spotting: DNN S-L | 0.82-0.86 (Acc) | 82-491 | 508-3029 |
| Keyword Spotting: CNN S-L | 0.91-0.92 (Acc) | 75-492 | 436-3029 |
| Keyword Spotting: MicroSpeech | 0.62 (Acc) | 18 | 112 |
| Image Classification: MobileNet v2 | 0.69 (Acc) | 3927 | 24215 |
| Anomaly Detection: MicroNet S-L | 0.95-0.96 (AUC) | 246-452 | 1523-2794 |

## 2.2 Part 2: Store, Execute ML from External FS

This part of OTA-TinyML enables storing of multiple ML models (fetched from a webserver) on any memory unit of choice. It is compatible with internal memory units like on-chip flash and SPI Flash FS (SPIFFS); Also external File Systems (FS) like EEPROM FS (EEFS), SD card FS (SDFS). This part also is responsible to load and execute models demanded by the IoT application. In conventional TinyML approaches, after the ML model training phase, the output model is converted to an array and exported as a C header file. This file is imported into the code of the IoT application (using `#include model_name.h`), on which the TF Lite Micro interpreter is run to obtain predictions via the EloquentTinyML Arduino Library [8].

Orthogonal to the conventional TinyML approaches, we show that, on the MCU boards, it is possible to load an ML model from a file instead of from a C array. During executing various TinyML models on MCU boards, we found out and also report here that *interpreters works identical in both cases - whether the model is declared as an array from the beginning or loaded as an array from somewhere else*. Using this finding/concept, OTA-TinyML initially reads a file (ML model in `model_name.bin` format stored in any memory unit) into a byte array. Then, it allocates memory for the read model using the `malloc()` function and copies the model content byte-by-byte, from the `.bin` file to the MCU SRAM memory, using which the interpreter produces predictions [9].

## 3 OTA-TINYML TESTING

The models and Arduino MCU boards used for OTA-TinyML testing are given in Table 1 (right). For part 1 testing, to ensure extensiveness, the `.h` file size of various ML models (from 112 KB MicroSpeech to 143421 KB Wav2Letter) are fetched from cloud to edge MCUs. For part 2 testing, 4 types of memory units are extensively used to test the onboard model storing and loading performance of OTA-TinyML. So, SDFS is interfaced to B3, B7; EEFS to B6, B4; Internal SPIFFS of B5; Internal flash memory of B1, B2. As shown in Figure 1, we first upload the `.bin` files of 16 pre-trained ML models (6 task types) into an HTTPS webserver. Then, the C++ implementation of the OTA-TinyML approach provided as a `.ino` file (server details entered in this file) is flashed on 7 different MCU boards B1 to B7 using Arduino IDE. At this stage, both the server and edge devices are ready for OTA-TinyML testing.

Starting from B1-B7, we instruct devices to initialize model fetching process. Boards B3, B7 with SDFS have highest storage capacity, so they downloaded all 16 models ($\approx$ 188 MB). Similarly, other boards downloaded models according to their memory limits. Next, the fetched model files get stored on the FS interfaced to the boards. Then, based on their SRAM capacity (see Table 1 (left)), models are loaded from FS and executed to produce inference results. For example, B4 with the least SRAM of 192 KB used OTA-TinyML only on MagicWand and MicroSpeech models. Whereas boards B1-B3, with better SRAM of 1 MB, used the OTA-TinyML to load and execute more model varieties. In summary, despite the diversity in MCU hardware specification or manufacturer, OTA-TinyML part 1 implementation, without stalling the devices, successfully fetched different size models from the cloud. The part 2 implementation successfully could store, load, execute models from internal memory (flash, SPIFFS) and also from external FS (SDFS, EEFS).

## 4 CONCLUSION

A resource-friendly OTA-TinyML approach was presented to enable end-to-end fetching, storage, and execution of many ML models on a single Arduino IoT board. OTA-TinyML provides developers and engineers the freedom to remotely re-purpose (load and run the model on demand) IoT devices on-the-fly without the need for physical reflashing. For example, with OTA-TinyML, even the 3$ ESP32 chip with only 4 MB flash can dynamically fetch *n* models from webserver such as keyword spotting (3MB), anomaly detection (2.7MB), visual wake words (3.2MB), etc., store in internal memory or external filesystems, then execute any ML model upon demand.

## ACKNOWLEDGEMENT

This publication has emanated from research supported in part by a research grant from SFI under Grant Number SFI/16/RC/3918 (Confirm) and also by a research grant from SFI under Grant Number SFI/12/RC/2289_P2 (Insight), with both grants co-funded by the European Regional Development Fund.

## REFERENCES

[1] M. M. Villegas et al. A study of over-the-air (ota) update systems for cps and iot operating systems. In *European Conference on Software Architecture*, 2019.

[2] K. Zandberg, E. Baccelli, et al. Secure firmware updates for constrained iot devices using open standards: a reality check. In *IEEE Access*, 2019.

[3] B. Sudharsan, P. Yadav, J. G. Breslin, and M. I. Ali. An sram optimized approach for constant memory consumption and ultra-fast execution of ml classifiers on tinyml hardware. In *IEEE Services Computing (SCC)*, 2021.

[4] B. Sudharsan, S. Salerno, D.-D. Nguyen, M. Yahya, A. Wahid, P. Yadav, J. G. Breslin, and M. I. Ali. Tinyml benchmark: executing fully connected neural networks on commodity microcontrollers. In *IEEE 7th WF-IoT*, 2021.

[5] B. Sudharsan, P. Patel, J. G. Breslin, and M. I. Ali. Enabling machine learning on the edge using sram conserving efficient neural networks execution approach. In *ECML PKDD*, 2021.

[6] N. Lethaby. A more secure and reliable ota update architecture for iot devices. In *Texas Instruments*, 2018.

[7] B. Sudharsan, R. Ranjan, et al. Ota-tinyml: over the air deployment of tinyml models and execution on iot devices. In *IEEE Internet Computing*, 2022.

[8] Eloquenttinyml: https://github.com/eloquentarduino/eloquenttinyml, 2021.

[9] B. Sudharsan and P. Patel. Machine learning meets internet of things: from theory to practice. In *ECML PKDD*, 2021.