Seminar Cloud Computing

From Concept to Production: Enablements TinyML in Industrial Setting

Trung Nguyen
Technische Universität München

November 2024

Abstract

In recent years, Artificial Intelligence (AI) and Machine Learning (ML) have received tremendous amount of attention in both industry and research world. However, conventional Machine Learning demands high computing capability which limits its usage to only larger computing units. The pardigm shift to Tiny Machine Learning (TinyML) is revolutionizing industries by enabling the deployment of machine learning models on low-power, resource-constrained devices. Being one of the most rapid developing field of Machine Learning, TinyML promises to benifits multiple industries. However, building a production-ready tinyML system poses different unique challenges. In this paper, we explore the key obstacles faced when developing and deploying TinyML models in production environments, including model optimization, hardware limitations, software integration, and maintaining performance in real-world conditions. Additionally, we present real-world use cases of TinyML in industrial settings, showcasing its transformative impact. We also discuss practical approaches and strategies presented by recent researches [11] to overcome these challenges, providing insights into how TinyML systems can be successfully scaled and implemented in production.

1 Introduction

Traditional Machine Learning Models, especially Deep Learning Models typically require substantial amount of computing capability to operate effectively. These models are often trained on powerful Graphics Processing Units (GPUs) and produce large models ranging from tens or hundreds of gigabytes (GB) down to smaller models in the range of 10 to 100 megabytes (MB). However, the memory requirements during runtime for these models far exceed what microcontrollers (MCUs) can handle. The pardigm shift to TinyML is driven by the prevailing number of Microcontroller Units (MCU) currently circulating in the industry. According to a recent report [2, 1], as of 2021, around 31 billion MCUs were shipped worldwide annually. The MCU market size is projected to increase in upcoming years [2]. This creates a big incentive for researchers and industry players to put effort into developing the technology further. TinyML aims to enable data processing or inferencing directly on embedded systems, particularly on Internet Of Things (IOT), instead of streaming to the cloud. TinyML models can operate on energy- and memory-constrained devices by minimizing communication with external servers, using optimized architecture designs, and employing compression techniques such as quantization and pruning. The advancement of tinyML has positively influenced multiple industries and sectors such as: industrial IOT [9], healthcare [4], agriculture [15], IOT in smart-city [6, 9].

Although various organizations have been actively investing resources into machine learning and data science projects, only around 13% of these projects successfully progress to production.[1] This low success rate highlights the complex challenges and gaps in translating ML concepts into fully functional, production-grade systems, especially in the realm of TinyML. In this paper, we aim to address these challenges and outline what is necessary to transition a TinyML project from concept to production, ultimately creating value for industrial applications. In Chapter 2, we provide an in-depth overview of the fundamental concepts, techniques, and development

[1] https://venturebeat.com/ai/why-do-87-of-data-science-projects-never-make-it-into-production/

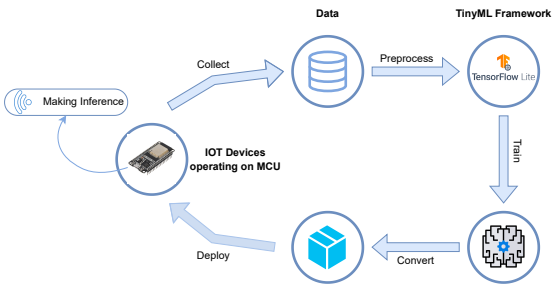

Figure 1: TinyML Deployment pipeline

pipeline of TinyML. Following this, in Chapter 3, we draw on recent research to investigate the specific challenges encountered in developing and deploying TinyML models in production environments, as well as practical strategies to overcome these obstacles. Next, in Chapter 4, we explore real-world applications of TinyML in industrial settings, illustrating its transformative potential. Finally, in Chapter 5, we consider the future trajectory of TinyML, discussing upcoming advancements and their implications for broader industrial adoption.

2 TinyML Overview

2.1 Key Concepts and Techniques

Pruning removes unnecessary neurons or connections from a neural network to reduce its size and complexity. This optimization lowers memory and computational demands, making the model faster and more efficient for deployment on resource-limited devices like microcontrollers.

Quantization reduces the precision of model parameters (e.g., from 32-bit to 8-bit), which decreases model size and computational needs. This technique enables efficient model inference on devices with limited memory and processing power, such as those used in TinyML applications.

With the TinyML model running on edge, the data is stored, processed and analysed internally rather than at an external server or cloud.

2.2 TinyML pipeline

Data Collection: The workflow starts with collecting data from an IoT device. This data serves as the input for training machine learning models. The IoT device can gather various types of data depending on the application, such as sensor data in smart homes or environmental monitoring.

Preprocessing: After collecting data, it moves to the preprocessing stage. Here, the data is cleaned and prepared for model training. Preprocessing can involve tasks such as normalization, handling missing values, or feature extraction to improve the quality of the input data.

Model Training: The preprocessed data is fed into an ML framework, where a machine learning model is trained. This stage involves using machine learning algorithms to find patterns in the data, building a predictive or analytical model that can generalize from the data.

Model Conversion: Once the model is trained, it is converted into a format suitable for deployment on resource-constrained devices, like microcontrollers. This step might involve techniques such as quantization (reducing the precision of model weights) or pruning (removing unnecessary parts of the model) to reduce the model's size and computation needs.

Model Packaging: After conversion, the model is packaged into a deployable form. This includes bundling the model with any necessary runtime components to allow it to be executed efficiently on the target device.

Deployment: The next step involves deploying the packaged model onto an IoT device. This means transferring the model to the device, setting it up for real-time inference or operation in the field.

Inference: Finally, the deployed model performs inference on the IoT device. Inference refers to the process of using the model to make predictions or decisions based on new data collected by the IoT device. The model operates locally on the device without needing continuous cloud connectivity, enabling real-time decision-making at the edge.

3 Enablements of TinyML in Industrial Setting

3.1 Challenges

The fundamental pipeline for deploying a TinyML model to a microcontroller, as described above in Chapter 2, provides a general framework for model development and deployment. However, in industrial settings, this pipeline often falls short due to a variety of challenges unique to production environments. Firstly, industrial applications typically demand more robust and adaptable systems, requiring continuous model updates and seamless integration within complex IoT infrastructures. Secondly, these demands introduce specific obstacles, such as the inability to regularly update models on resource-constrained devices and the difficulty of integrating TinyML systems with existing IoT architectures.

3.1.1 Adaptation to unseen scenarios and constantly changing conditions

In traditional machine learning (ML), models can be regularly updated with new data through a constant internet connection, allowing them to stay relevant as conditions evolve. This capability, however, is often unavailable for TinyML models deployed on microcontroller units (MCUs) in IoT or edge devices, which typically lack continuous internet access. Consequently, once a model is deployed on an MCU, it may remain static for extended periods, operating without the benefit of regular updates.

This static deployment significantly impacts TinyML models over time, especially in dynamic environments where data patterns evolve. In real-world applications, conditions rarely remain constant, and the input data the model receives can change due to factors such as seasonal variations, environmental shifts, or user behavior changes. Without regular updates, a static model can struggle to adapt to these evolving patterns, a challenge known as concept drift (when the relationship between input and output changes) and data drift (when the distribution of input data changes). Both forms of drift can lead to a decline in model performance, making it essential to consider ways to address these challenges in TinyML deployments.

3.1.2 Integrating with Existing IOT System and Management of TinyML Models in Hetegenous Devices

Integrating TinyML-enabled microcontrollers (MCUs) into existing IoT systems presents several unique challenges. IoT infrastructures are often built on established architectures, protocols, and data formats that may not be directly compatible with TinyML MCUs. Adapting these MCUs to work within the existing IoT ecosystem requires significant customization to ensure seamless data exchange, communication, and interoperability.

Moreover, existing IoT systems frequently rely on centralized processing or cloud-based analytics, while TinyML emphasizes local computation on edge devices. This architectural shift introduces complexities in synchronizing data and managing hybrid workflows that combine cloud and edge processing. Industrial IoT deployments also come with strict security and compliance requirements, so integrating TinyML MCUs necessitates additional security protocols, device authentication, and data encryption to maintain data integrity across the IoT network.

Maintenance and scalability issues can complicate the integration of TinyML into IoT systems at scale. Updating models on distributed MCUs, managing firmware compatibility, and monitoring device health require robust infrastructure and automated processes, adding layers of complexity to deploying and managing TinyML solutions within existing IoT frameworks. Moreover, to manage and operate such sophisitcated system requires deep technical capability which many of professionals with good domain lacks. This urges not only the need for effective methods to manage and maintain TinyML systems at scale but also the importance of making these systems accessible to a broader range of professionals.Enabling more individuals to work with TinyML systems will be crucial for ensuring widespread adoption and operational success in industrial environments. [6, 8, 5, 13, 14].

3.2 Bringing tinyML to production environment

In order to target each of the outlined challenges, we have found in recent researches the following approaches proposed by Ren et al. [11, 12, 10].

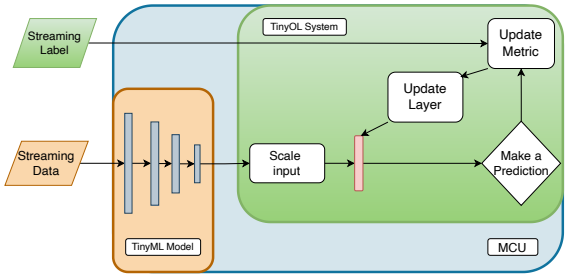

Figure 2: TinyOL Components on MCUs (adapted from Ren et al.)

3.2.1 Advanced On Device Learning Methods

Creating a general NN model for general use is impractical for MCUs due to their resource-constraint nature. A simple NN model can easily exceed the hard storage room of a MCU. Moreover, for a general NN model to function, it needs to be updated with field data on the fly, which usually not possible as MCUs do not possess constant connection to the internets, especially while operating at edge.

In their paper on TinyOL (Tiny Machine Learning with Online Learning), Ren et al. introduced a novel system that enables on-device learning. This approach allows models deployed on embedded devices, such as sensors or microcontrollers, to learn and adapt in real time without requiring external retraining.

The core of TinyOL lies in the additional layer marked in red in Figure 2 reffered to as the **update layer** which consists of neurons that can be customized, initialied, and updated on the fly. This layer serves as an output layer that can adapt to new data while preserving the original network's structure. Following is the breakdown of the other components in Figure 2.

- **Streaming Data**: This component represents the live data input collected by the MCU from its environment. It is fed into the TinyML model continuously, allowing the system to make predictions based on real-time information.
- **TinyML Model**: The core machine learning model deployed on the MCU. It takes streaming data as input and performs initial processing. This model consists of several layers (depicted by vertical bars), which extract features from the data.
- **Scale Input**: After the TinyML model processes the input data, it passes through a scaling component. This step standardizes or normalizes the data to ensure it fits the model's expected input distribution, helping maintain consistent performance as new data arrives.
- **Make a Prediction**: After scaling and processing, the TinyOL system uses the model to make predictions. This decision node represents the point at which the system outputs a result based on the latest input data.
- **Update Metric**: When true labels (ground truth) are available, the system can update evaluation metrics based on the prediction's accuracy. This component helps measure model performance and provides feedback for updating the model's parameters.
- **Streaming Label**: This component represents the true labels or ground truth data available intermittently. When available, it is used to calculate metrics and adjust the update layer, enabling the model to correct itself over time and adapt to changing data patterns.

TinyOL Workflow The **TinyOL workflow** is summarized in the authors' algorithm [11]. The workflow proceeds as follows:

1. *Initialization:* TinyOL initializes both the TinyML model and the TinyOL system for on-device learning.
2. *Streaming Data Processing:* As new data points are received from the data stream, TinyOL processes each sample iteratively.
3. *Normalization and Standardization:* The running mean and variance of the incoming data are updated for standardization purposes, preparing the data for the model.
4. *Prediction and Model Update:* TinyOL uses the standardized input to make a prediction. If a true label is available, TinyOL computes the error between the prediction and the label, updating evaluation metrics. The model's weights are then adjusted accordingly using techniques such as **stochastic gradient descent (SGD)**.

Minium Resource Consumption: After processing each data point, TinyOL discards it to conserve memory. Compared to traditional batch/offline training methods, this efficient approach enables

TinyOL to handle high volumes of streaming data, which is essential for resource-constrained MCUs with limited storage and processing capacity. The framework operates with as little as 7 KB of RAM and 135 KB of Flash memory.

Continuous Adaptation: TinyOL's architecture allows it to continuously adapt to new data patterns over time, addressing issues like **concept drift** and **data drift**. By updating model weights in real time, the system ensures that the deployed model remains relevant and can handle changes in input data characteristics that may not have been present during initial training.

3.2.2 Management of TinyML systems at scale

There has been increased attention to TinyML Management with differents methods proposed such as: TinyML as a service , Model Card to describe and organize the information of trained models [7], or TinyML Operations (TinyMLOps) [3] to build an abstraction layer for various hardware compilers. However, theses approaches reveal certain drawbacks as they lack features for users to store and exchange information about existing resources.

The paper introduces a TinyML low-code management framework called SeLoC-ML (Semantic Low-Code ML), designed to simplify the deployment, management, and scaling of TinyML applications. The essentials of SeLoC-ML can be broken down into two main parts: to build a knowlegde graph ontology on the existing overal TinyML system and to inject the knowlegde graph into a low-code platform [2].

Contruction of the knowlegde graph By reusing existing standards, like the **Semantic Sensor Network(SSN)** and **W3C Thing Description (TD)** [3] ontologies (a structured schema), an ontology that includes defitions and metadata for TinyML models, such as neural network architecture, layer configurations, and hardware requirements can be built. These data then can be stored using a graph database to facilitate management, dicovery, and deployment of TinyML applications.

[2]A low-code platform is a development environment that enables application creation through visual interfaces and minimal coding, making it accessible to users with limited programming knowledge. `https://www.ibm.com/topics/low-code/`

[3]`https://www.w3.org/WoT/`

example: The Move model, as depicted in the knowledge graph, represents a TinyML motion-detection or activity-recognition model with defined input and output requirements, metrics, and compatibility conditions. By including this model in the knowledge graph, the TinyML management system can ensure the Move model is deployed on compatible hardware, facilitating real-time motion analysis or gesture detection in various applications.

Integrating into a low-code platform integrates with Mendix, a low-code platform, to enable non-experts to manage and deploy TinyML models across diverse embedded devices.

Result This approach addresses the challenges of **updating models on distributed MCUs** and **managing firmware compatibility** by leveraging Semantic Web technologies and a Knowledge Graph (KG) to centralize and standardize information about TinyML models and IoT devices. The framework's *ontology* provides a standardized way to describe ML models and IoT devices, facilitating tracking of model versions, configurations, and updates across various devices. By storing information about model updates, dependencies, and compatibility requirements in the KG, the system enables automated discovery of compatible models for each device, allowing administrators to push updates more efficiently. The *querying capabilities* of the Knowledge Graph make it easy to identify which MCUs require updates, simplifying model deployment across large-scale, distributed systems. Additionally, the KG maintains details about each device's firmware, hardware specifications, and compatibility requirements, ensuring that only compatible firmware and model updates are deployed. This centralized approach minimizes the risk of deploying incompatible firmware or models, which is crucial in environments with diverse MCU configurations.

4 Use Cases of TinyML in Industrial Setting

Enumerations using bullet points:

- Agriculture
- Environmental Monitoring
- Industrial predictive maintenance
- Edge AI and Autonomous Systems

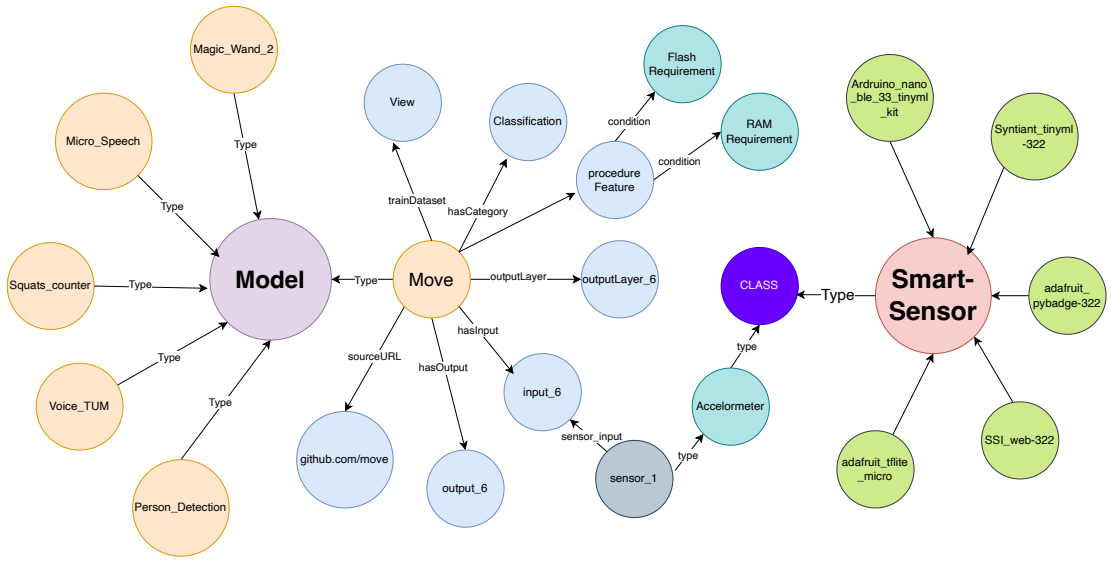

Figure 3: Visualization of System's Knowledge Graph

5 Future of TinyML

6 Conclusion

References

[1] Microcontroller market research, 2030.

[2] Microcontroller Market Size, Share & Trends By 2034.

[3] Mattia Antonini, Miguel Pincheira, Massimo Vecchio, and Fabio Antonelli. Tiny-MLOps: a framework for orchestrating ML applications at the far edge of IoT systems. pages 1–8, May 2022. ISSN: 2473-4691.

[4] Mamta Bhamare, Pradnya V. Kulkarni, Rashmi Rane, Sarika Bobde, and Ruhi Patankar. Chapter 14 - TinyML applications and use cases for healthcare. pages 331–353, January 2024.

[5] Miguel de Prado, Manuele Rusci, Romain Donze, Alessandro Capotondi, Serge Monnerat, Luca Benini And, and Nuria Pazos. Robustifying the Deployment of tinyML Models for Autonomous mini-vehicles, July 2020.

[6] Dina Hussein, Dina Ibrahim, and Norah Alajlan. Original Research Article TinyML: Adopting tiny machine learning in smart cities. *Journal of Autonomous Intelligence*, 7:1–14, January 2024.

[7] Andrew Zaldivar Margaret Mitchell, Simone Wu. Model Cards for Model Reporting. 2019.

[8] Aditya Jyoti Paul, Puranjay Mohan, and Stuti Sehgal. Rethinking Generalization in American Sign Language Prediction for Edge Devices with Extremely Low Memory Footprint, February 2021. arXiv:2011.13741.

[9] Partha Pratim Ray. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623, April 2022.

[10] Haoyu Ren, Darko Anicic, Xue Li, and Thomas Runkler. On-device Online Learning and Semantic Management of TinyML Systems. *ACM Transactions on Embedded Computing Systems*, 23(4):1–32, July 2024.

[11] Haoyu Ren, Darko Anicic, and Thomas Runkler. TinyOL: TinyML with Online-Learning on Microcontrollers. April 2021. arXiv:2103.08295.

[12] Haoyu Ren, Darko Anicic, and Thomas Runkler. How to Manage Tiny Machine Learning at Scale: An Industrial Perspective. February 2022. arXiv:2202.09113.

[13] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. The synergy of complex event processing and tiny machine learning in industrial IoT. In *Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems*,

DEBS '21, pages 126–135, New York, NY, USA, June 2021. Association for Computing Machinery.

[14] A. Navaas Roshan, B. Gokulapriyan, C. Siddarth, and Priyanka Kokil. Adaptive Traffic Control With TinyML. In *2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 451–455, March 2021.

[15] Vasileios Tsoukas, Anargyros Gkogkidis, and Athanasios Kakarountas. A TinyML-based System For Smart Agriculture. pages 207–212, November 2022.