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 [8] 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 [7], healthcare [3], agriculture [11], IOT in smart-city [5, 7].

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/

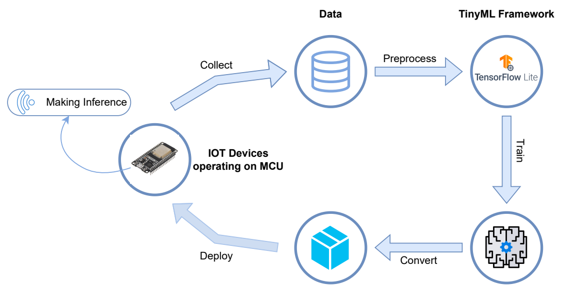

Figure 1: TinyML Deployment pipeline

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.

This urges the need for a sufficient method to manage and maintain TinyML system at scale.

To tackle this, we have found approaches from Paper A, Paper B.

[5, 6, 4, 9, 10].

3.2 Bringing tinyML to production environment

3.2.1 Advanced On Device Learning Methods

TinyOL (Tiny Machine Learning with Online Learning), as presented in the paper, is a framework designed to enable on-device learning. This means it allows models deployed on embedded devices (like sensors or microcontrollers) to learn and adapt in real-time without needing to be retrained externally. Here's how it works and why it does not require an internet

On-device Incremental Learning TinyOL enables continuous learning directly on resource-constrained devices. It can train a model incrementally by learning from streaming data as it arrives in real-time, without the need to store large datasets. This approach allows the model to remain updated with new data without requiring frequent retraining in centralized, powerful systems.

Adapt TinyOL is particularly useful in cases where the environment is dynamic and subject to changes. As new data arrives, the model adjusts incrementally, making it more robust to "concept drift" (changes in data patterns over time) without requiring access to previous data. This is critical in real-world scenarios where conditions may fluctuate frequently.

Minimal Resource consumption TinyOL operates with minimal memory and computational resources. For instance, the framework can run within just 7 KB of RAM and 135 KB of Flash memory, making it suitable for low-power embedded devices that cannot support traditional machine learning models.

How TinyOL Works

1. The device collects streaming data from its environment.
2. TinyOL processes the data, updates the model's running mean and variance, and adjusts the model weights accordingly.
3. After processing each new data point, the model discards the data, ensuring efficient memory usage.
4. The model is continuously updated, allowing it to handle shifts in data patterns over time.

3.2.2 Low-code management system

MLOps for Scaling TinyML: TinyMLOps has some drawbacks

The paper introduces a TinyML low-code management system called SeLoC-ML, designed to simplify the deployment, management, and scaling of TinyML applications. This system is built on semantic web technology and integrates with Mendix, a low-code platform, to enable non-experts to manage and deploy TinyML models across diverse embedded devices.

Workflow of SeLoC-ML

1. Define Requirements: Users specify the model requirements or device specifications (e.g., sensor types, memory limits) via the Mendix interface.
2. Match Models and Devices: The system queries the KG for matching models and devices based on the user's input. The system ensures that the selected models meet the hardware constraints of the device.
3. Generate Deployment Code: Once a model and device are matched, SeLoC-ML generates a deployment project, including configuration files and scripts, which are ready to be uploaded to the target hardware. This eliminates the need for manual coding, reducing development time and errors.
4. Deploy and Manage at Scale: SeLoC-ML supports large-scale management, allowing users to track, update, and maintain TinyML models across multiple devices.

4 Use Cases of TinyML in Industrial Setting

Enumerations using bullet points:

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

5 Future of TinyML

6 Conclusion

References

[1] Microcontroller market research, 2030.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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