12

A TinyMLOps Framework for Real-world Applications

Mattia Antonini, Massimo Vecchio, and Fabio Antonelli

Fondazione Bruno Kessler, Italy

Abstract

As devices become smarter, embedding intelligence in microcontrollers and constrained environments is critical. Optimising machine learning models for these tiny devices requires balancing software efficiency, such as accuracy, with hardware constraints like memory and power. We introduce a TinyMLOps-based framework for optimising models across the cloud-to-device continuum. In our approach, cloud resources handle heavy tasks like data labelling and model training, while microcontrollers gather real-time metrics on efficiency and hardware utilisation. Then, some repositories manage models and metadata identified during the optimisation phase, including performance metrics collected directly from the target devices, thus ensuring an accurate exploration of the model space in real-world conditions. Using tools such as MicroPython and MLFlow, our framework enables seamless AI deployment on resource-constrained devices, providing a scalable solution for the future of edge AI.

Keywords: MLOps, edge AI, edge computing, model deployment, AI workflow, real-time metrics.

12.1 Introduction

Artificial intelligence (AI) is becoming an invaluable companion in everyday life, present in wearables, smartphones, cars, and homes. We use AI to

write, compose music, and edit pictures, making access almost effortless. However, this can obscure the real challenges of deploying AI models, especially on devices at the edge of the network [1]. These devices include single-board computers (e.g., Raspberry Pi, Coral Dev Kit, NVIDIA Jetson boards), which support full-fledged operating systems (OS) like Linux but are costly and energy-intensive. In contrast, microcontrollers (MCUs) offer lower computational power but are significantly cheaper and less demanding regarding energy and hardware resources. MCUs are ideal for specialised, real-time tasks in IoT environments but require adapted methodologies for their management [2]. The absence of a full-fledged OS and the constrained computational environment demand adjustments in orchestrating AI workflows on such devices.

In this context, the TinyMLOps methodology presented in [3] can be fundamental in designing, deploying, and monitoring AI capabilities on constrained devices. From an operational standpoint, US-based companies like Roboflow [4], Edge Impulse [5], and Neuton.ai [6] provide platforms for designing, optimising, deploying, and executing AI pipelines at the network edge. While Roboflow focuses on computer vision problems, primarily supporting single-board computers, Edge Impulse specialises in creating and deploying machine learning models on resource-constrained devices like MCUs, making it ideal for IoT applications. However, it requires hardcoding models into firmware, updated via Over-the-Air (OTA) updates, which reduces flexibility in real-time scenarios. Similarly, the Neuton TinyML platform, provided by Neuton.ai, leverages a no-code approach to generate ultra-compact neural networks, optimising resource use for constrained devices. Its patented framework incrementally builds neural networks neuron by neuron, resulting in models significantly smaller than those from other frameworks. It enables deployment on devices with as little as 8-bit capacity, making it highly suitable for diverse IoT applications.

This paper presents a TinyMLOps framework architecture designed to streamline AI workflows on MCUs, enabling seamless model deployment without embedding the model in the firmware. We describe the framework's components, their role in supporting the methodology, and the various actors involved, from data labelling to model design, optimisation, firmware engineering, and operations. Furthermore, we provide a set of candidate state-of-the-art technologies that can be adopted to implement the framework and demonstrate its real-world feasibility.

The rest of the paper is structured as follows. Section 12.2 gives an overview of the TinyMLOps methodology, then Section 12.3 presents the

framework architecture by describing all the components. Section 12.4 provides a technological landscape for the framework. Finally, Section 12.5 concludes the paper.

12.2 TinyMLOps methodology

The TinyMLOps methodology [3] evolves the MLOps methodology [7] to bring AI workflows and models to the edge and far edge of the network.

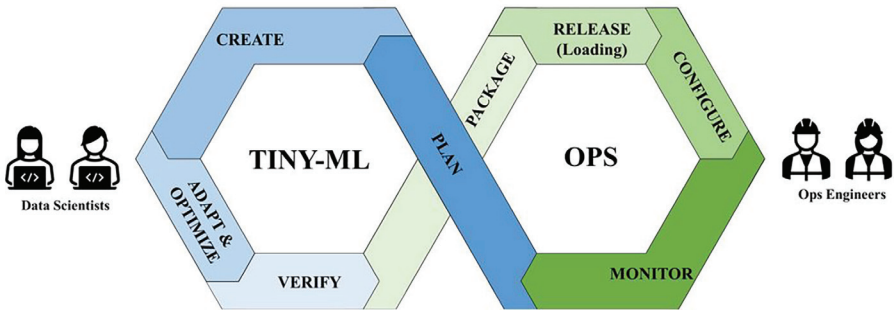

Figure 12.1 The TinyMLOps Loop [3].

TinyMLOps particularly tackles the adaptation, deployment, and monitoring of models even on low-end MCUs (e.g., Raspberry Pi Pico, ESP32, Arduino, and STM32 families).

The TinyMLOps approach is illustrated as a simple loop, as depicted in Figure 12.1.

In more detail, the TinyMLOps loop starts with the **TinyML circle** (on the left side, highlighted in blue, Figure 12.1). It proceeds through a series of iterative steps involving collaboration between data scientists and operations engineers (on the right side highlighted in green in the Figure 12.1) to optimise and deploy AI models on edge devices. The steps are briefly introduced as follows.

- **Plan**: data exploration and feature engineering to prepare and optimise data for model development;
- **Create**: model training and feature adaptation tailored to the problem space;
- **Adapt & Optimize**: the trained model is adapted and optimised for the target computing platform, typically through techniques such as quantisation and pruning;

- **Verify**: ensures model compatibility and executability on the target platform, confirming that all operations are supported;
- **Package**: the model is compiled or converted into a deployable format (e.g., TFLite) suitable for execution on edge devices;
- **Release**: the model is deployed into the device memory, making it ready for inference;
- **Configure**: run-time parameters may be adjusted based on platform- and application-specific requirements (e.g., detection threshold);
- **Monitor**: ongoing real-time surveillance of the model's performance and health, ensuring its reliability post-deployment. If the model behaves unexpectedly, it triggers a new iteration of the entire loop.

This closed-loop process facilitates seamless adaptation, deployment, and monitoring of TinyML models, ensuring efficient execution on resource-constrained edge devices.

12.3 A TinyMLOps framework architecture

As discussed above, the TinyMLOps methodology encompasses multiple abstract phases to deliver an efficient pipeline for deploying and maintaining machine learning models on edge devices. To implement these phases in real-world applications, we need to translate them into concrete components and pipelines within a software framework architecture. Figure 12.2 illustrates this architecture, which is divided into two sections corresponding to the two circles of the TinyMLOps loop (Figure 12.1). The left side (in blue) represents the components required for handling the TinyML-specific tasks, while the right side (in green) focuses on the components involved in the methodology's operations phase.

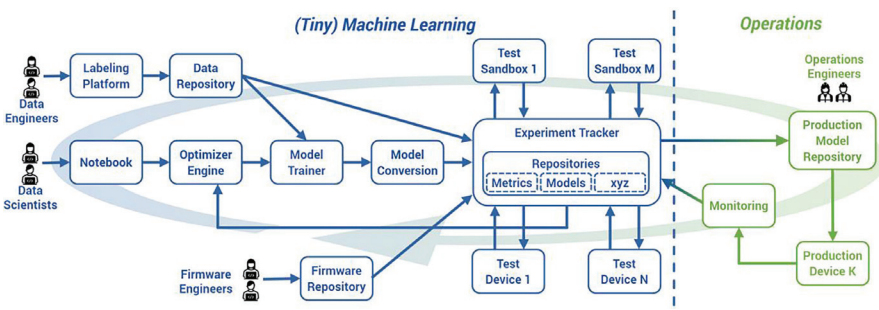

Figure 12.2 TinyMLOps Framework Architecture.

Starting from the TinyML part, we can distinguish three main pipelines: the firmware engineering pipeline, the data engineering pipeline, and the data science pipeline, which are introduced as follows:

- **firmware engineering pipeline**: managed by firmware engineers, produces the various firmware versions for the edge devices supported by our framework. These firmware binaries are organised and stored in the Firmware Repository, ready for deployment. For example, one firmware might be based on MicroPython[1], compiled for ESP32 and equipped with libraries tailored to the target application.
- **data engineering pipeline**: managed by data engineers, provides a UI through the Labelling Platform for labelling data and storing the information (e.g., labels, metadata) in the Data Repository. This repository plays a crucial role, as it is accessed during model training and validation.
- **data science pipeline**: managed by data scientists, which offers the possibility to design, develop, and test models, as detailed below.

Models are initially created using a Notebook environment (e.g., Jupyter Notebooks, VSCode) where scientists can define the computing pipeline, the model architecture, and the relevant hyperparameters ranges based on the target metrics they aim to optimise (e.g., model size vs. accuracy). Additionally, they can specify the desired hardware platform for deployment, ensuring the model aligns with the constraints and capabilities of the chosen edge device.

At this stage, the Optimization Engine generates and manages various parameter combinations to explore and optimise the search space. In our case, this process is handled through a metaheuristic approach, ensuring effective exploration and solution space optimisation. This includes not only the hyperparameters of the models but also factors such as hardware configurations, allowing for an optimal balance between performance metrics (e.g., model size, accuracy, latency) and resource constraints.

The combination of parameters and the model definition (e.g., code) is passed to the Model Trainer component, which trains the model using data fetched from the Data Repository. However, the trained model may not yet be fully compatible with the target hardware platform. To address this, the model is further processed by the Model Conversion component, which applies various techniques such as quantisation, pruning, and other optimisations to ensure the model is adapted for efficient execution on the specific hardware platform.

[1] https://micropython.org/

At this stage, it is essential to evaluate the model's performance. To efficiently manage the artefacts (i.e., model binary files), along with associated parameters (e.g., data used to train and model hyperparameters) and metrics, we rely on the Experiment Tracker to handle the storage of data and metadata in dedicated internal repositories, accessible via APIs.

Based on the metrics we aim to measure and the target hardware platform, the model can be deployed in the appropriate computing environment: a cloud-based Test Sandbox for software metrics (e.g., accuracy) and physical Test Devices for hardware-specific metrics (e.g., latency, memory usage). This approach ensures a clear understanding of the model's performance in real-world environments, as testing is carried out on actual computing units. This provides insights into software and hardware efficiency, allowing for more accurate metrics assessments.

The collected metrics are then returned to the Optimizer Engine, where they are used to further refine the hyperparameters, model configuration, and hardware selection by triggering new iterations.

This continuous feedback loop allows the system to iteratively improve the model, ensuring optimal performance and deployment on the target edge devices. All tested models, along with their associated parameters and metrics, are stored in the Experiment Tracker. This comprehensive tracking provides valuable insights into how models can be further optimised for the specific application and platform. This is beneficial as it allows for easy access, tracking, and comparison of different model iterations, ensuring that the most effective model can be quickly identified and deployed. Additionally, this centralised storage facilitates collaboration between teams, improves reproducibility, and supports ongoing optimisation efforts by making historical data readily available for analysis.

Once the best model is identified, fully optimised, and validated, it is promoted from the Model Repository inside the Experiment Tracker to the Production Model Repository, making it ready for deployment on production devices. This transition marks the shift from development and testing to the operational phase, managed by Operations Engineers. From the Production Model Repository, the model is deployed to Production Devices, typically (far-)edge devices operating in a live environment. Operations Engineers oversee this deployment, ensuring the model performs reliably and efficiently in real-world conditions.

Throughout the Operations phase, continuous monitoring is crucial. Operations Engineers may continue to observe the model's performance, gathering insights such as real-time metrics, latency, and memory usage using

the Monitoring component. If any issues arise or further optimisations are needed, the model can be updated or refined through subsequent iterations, with performance feedback potentially being routed back into the development process to trigger new rounds of optimisation and improvements via the Experiment Tracker. This ongoing cycle ensures that the model remains efficient and effective as it operates on resource-constrained edge devices, seamlessly fulfilling the principles of the TinyMLOps methodology.

12.4 Technology Overview for TinyMLOps Adoption

Adopting the TinyMLOps methodology into real-world scenarios requires integrating a wide range of cutting-edge technologies that allow us to tackle the different challenges of deploying machine learning models on resource-constrained devices in the edge and far-edge of the network. The current landscape, driven by open-source innovation, encompasses platforms and tools that ease data management and versioning, model design, development, optimisation, and monitoring in environments where computational power and storage are limited.

Starting with the **firmware engineering pipeline**, several tools are available to support the early-stage development and production deployment of machine learning models on hardware-constrained devices. On the hardware side, embedded platforms such as Arduino, ESP32, Raspberry Pi Pico, ARM, and STM32 are widely adopted. These platforms are typically paired with development environments like Mbed OS, PlatformIO, ESP-IDF, and STM32Cube, which aid in firmware development, sensor integration, and more. For software, frameworks like TensorFlow Lite for Microcontrollers [8] and pure C/C++ code (e.g., code generated by STM32Cube.AI), along with a Real-Time Operating System (e.g., FreeRTOS), simplify model deployment on embedded devices. However, since the model is embedded within the device's firmware, updating it typically requires a complete flash. On the same side, MicroPython offers an alternative solution by enabling the creation of firmware that exposes a Python interpreter running on the constrained device, effectively functioning as a Hardware Abstraction Layer (HAL)[9]. This approach decouples the machine learning model from the underlying hardware and dependency libraries, allowing for the deployment of only a Python script and associated binary files without the need for a full firmware flash. The device can initiate updates through a simple HTTP request, significantly streamlining the process and making it more flexible and efficient.

For the **data engineering pipeline**, tools like Label Studio, Universal Data Tool, or Computer Vision Annotation Tool (CVAT) are critical for labelling and annotating complex datasets, ranging from images to time-series sensor data. These labelled datasets are managed using tools like Pachyderm, which combines scalable data storage with version control, making it ideal for tracking large volumes of data throughout different iterations of the machine-learning pipeline. Object storage solutions such as MinIO or AWS S3 are often employed to store these datasets, ensuring scalability and accessibility during model development with a simple and standardised API. Alternatively, it is possible to integrate, via APIs, existing platforms like Edge Impulse or Roboflow to exploit their data labelling functionalities. Based on cost constraints, privacy, and use-case requirements, it is possible to choose one solution or the other.

Lastly, the **data science pipeline** relies massively on cloud-based tools to design, develop, evaluate, and optimise machine learning models. Kubeflow, a powerful open-source framework, is crucial in orchestrating machine learning workflows, from data preprocessing to model training. By leveraging Kubernetes for scalable and efficient workload management, Kubeflow standardises the MLOps process, streamlining the entire model development lifecycle. It natively integrates with platforms such as Jupyter Notebooks and VSCode, providing collaborative environments for interactive model development and pipeline processing. A core feature of Kubeflow is its Pipelines component, which automates end-to-end workflows by structuring them as Directed Acyclic Graphs (DAGs). Each pipeline component is encapsulated in a Docker container, ensuring modularity and enabling integration with various tools and libraries. These pipeline steps can include tasks such as data preprocessing, model training, and evaluation.

A key step often incorporated is hyperparameter optimisation, which determines the best hyperparameter configuration for a given model. Katib, a component of Kubeflow, streamlines this process by supporting both standard and advanced search techniques, ensuring optimal performance for the model and hardware platform. This approach allows for co-design between software and hardware, optimising both simultaneously to target the deployment environment. This stage also often involves conversion tools to export models in TensorFlow Lite and ONNX (Open Neural Network Exchange) formats to enable optimised evaluations for edge devices. Such models can be stored in an Experiment Tracker, like the MLFlow Tracking, along with their metadata to study and supervise the optimisation process properly.

Before the deployment in production environments, models are evaluated in Test Sandboxes using Docker-based technologies like Seldon Core, KServe, or BentoML. For tests on Test Devices, models can be deployed using OTA updates by replacing the firmware or using MLFlow Tracking APIs by downloading models into device flash memory. Performance metrics are logged in the Experiment Tracker (MLFlow Tracking), enabling a feedback loop for subsequent iterations of the model optimisation. MLFlow also offers the Model Registry, a repository for models that have reached maturity for production deployment. Using some internal labels, it is possible to tag models for the target device or deployment scenario. Then, the model can be deployed via OTA updates or via the MLFlow APIs.

Finally, tools like Prometheus and Grafana enable real-time monitoring of the production devices, providing detailed insights into hardware (latency, memory usage, and CPU load) and software performance metrics (errors, crashes, anomalies). Prometheus collects and stores time-series data from the devices, while Grafana visualises this information through customisable dashboards by also enabling webhooks to trigger new optimisation iterations automatically.

This integration allows operations teams to proactively track the health of models and devices, ensuring that any performance issues can be addressed promptly to maintain optimal efficiency and reliability in the deployed system.

12.5 Conclusions

The ubiquitous presence of AI in everyday aspects of our lives is deeply transforming how we approach the lifecycle of models. From the initial model design to the deployment on production devices, multiple steps must be completed, many of which are often taken for granted. Every phase (e.g., data collection, data labelling, model training, model optimisation) involves complex processes that need to be carefully orchestrated to ensure optimal model performance in real-world scenarios.

In these settings, the TinyMLOps methodology clearly defines the high-level steps that must be tackled. Building on this foundation, we propose a software framework architecture streamlining these processes along the cloud-to-thing continuum. Our framework enables the design, development, and deployment of machine learning models while managing hardware constraints effectively. By integrating real-world tools like MicroPython

for lightweight and simple HAL and MLFlow for experiment tracking and deployment, the framework addresses key challenges in scaling AI applications to tiny devices, such as resource management and deployment complexity. Additionally, it enhances adaptability and scalability across diverse platforms, ensuring that models perform efficiently even in the most constrained environments.

Acknowledgments

This paper was partially supported by the Interconnected Nord-Est Innovation Ecosystem (iNEST) and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR)—MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.5—D.D. 1058 23/06/2022, ECS00000043). Moreover, it was partially supported by the Horizon Europe "NEUROKIT2E" project (Grant Agreement 101112268) and by the Horizon Europe "Arrowhead fPVN" project (Grant Agreement 101111977).

References

[1] T. Meuser et al., "Revisiting Edge AI: Opportunities and Challenges," IEEE Internet Comput., vol. 28, no. 4, pp. 49–59, Jul. 2024, https://doi.org/10.1109/MIC.2024.3383758.

[2] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "An Adaptable and Unsupervised TinyML Anomaly Detection System for Extreme Industrial Environments," Sensors, vol. 23, no. 4, p. 2344, Feb. 2023, https://doi.org/10.3390/s23042344.

[3] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "Tiny-MLOps: a framework for orchestrating ML applications at the far edge of IoT systems," in 2022 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS), Larnaca, Cyprus: IEEE, May 2022, pp. 1–8. https://doi.org/10.1109/EAIS51927.2022.9787703.

[4] Dwyer, B., Nelson, J., Hansen, T., and et. al., Roboflow (Version 1.0). (2024).

[5] S. Hymel et al., "Edge Impulse: An MLOps Platform for Tiny Machine Learning," 2022, arXiv. https://doi.org/10.48550/ARXIV.2212.03332.

[6] "Neuton.AI - No-code artificial intelligence for all." Accessed: Jan. 10, 2025. [Online]. Available: https://neuton.ai/

[7] M. M. John, H. H. Olsson, and J. Bosch, "Towards MLOps: A Framework and Maturity Model," in 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Palermo, Italy: IEEE, Sep. 2021, pp. 1–8. https://doi.org/10.1109/SEAA53835.2021.00050.

[8] R. David et al., "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," 2020, arXiv. https://doi.org/10.48550/ARXIV.2010.08678.

[9] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "A TinyML approach to non-repudiable anomaly detection in extreme industrial environments," in 2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), Trento, Italy: IEEE, Jun. 2022, pp. 397–402. https://doi.org/10.1109/MetroInd4.0IoT54413.2022.9831517.