TinyLearnED: Adaptive On-Device Edge Learning for IoT Anomaly Detection

ABDULAZIZ ALJOHANI, Cardiff University, UK and King Abdulaziz University, Saudi Arabia
HAKAN KAYAN, Cardiff University, UK
OMER RANA, Cardiff University, UK
CHARITH PERERA, Cardiff University, UK

Anomaly detection in cyber-physical systems (CPS) is difficult because system behaviour changes over time, and new attacks can appear after deployment. Most TinyML-based anomaly detection systems rely on static, pre-trained models, which cannot adapt once deployed. In addition, ground-truth labels are often unavailable during real CPS operations, restricting effective online learning. We present TinyLearnED, a weakly supervised framework for online anomaly detection. The framework supports online adaptation during the evaluated data streams without requiring post-deployment ground-truth labels. In particular, the framework exploits correlations among independent edge devices that observe the same physical environment. By exchanging binary anomaly decisions, these devices collaboratively generate pseudo-labels for training. Based on these pseudo-labels, each device performs local online fine-tuning, while a lightweight federated aggregation step averages final-layer parameters to smooth updates and improve resilience under the evaluated noisy and compromised-node conditions. During this process, the pre-trained feature extractor remains unchanged, and only a lightweight output layer is fine-tuned. We evaluate TinyLearnED on four XIAO ESP32S3 monitoring nodes using chronological replay of the Bolt 3D dataset, collected from one Prusa MK4 printer producing a single bolt geometry with triaxial accelerometer sensing. The results demonstrate that on-device adaptation is feasible without post-deployment ground-truth labels and yields task-dependent detection improvements in this testbed. Cross-geometry transfer and long-term deployment stability remain unevaluated.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; *Architectures*; • **Security and privacy → Hardware attacks and countermeasures**;

Additional Key Words and Phrases: Internet of Things, federated learning, anomaly detection, TinyML, online machine learning

1 Introduction

Anomaly detection in cyber-physical systems (CPS) presents significant challenges due to the dynamic nature of operating conditions over time. Traditional models trained offline operate under the assumption that normal and abnormal patterns remain static, an assumption that fails to hold in practice where concept drift and unforeseen anomalies frequently emerge. Cloud-based retraining approaches suffer from substantial latency, high costs, and feasibility constraints, particularly when devices operate with limited connectivity. This motivates the need for online machine learning, enabling systems to learn on-device from recent data and respond immediately to environmental changes. However, achieving reliable online adaptation on resource-constrained devices remains challenging, as ground-truth labels are rarely available post-deployment and edge devices face computational and memory constraints. These limitations prevent practical use in dynamic CPS environments.

Unfortunately, existing on-device online learning frameworks are not designed to adapt reliably in dynamic CPS environments without labels. LifeLearner [16] relies on a labelled replay buffer and TinyTL [9] assumes ground-truth labels remain available post-deployment, while TinyOL [28] updates a small set of parameters online but provides no mechanism to obtain a reliable per-window update signal in the field, where blindly adapting to recent data can absorb anomalies or spoofed measurements into the model. Our evaluation (Section 7.2) quantifies this empirically

Authors' Contact Information: Abdulaziz Aljohani, aljohania1@cardiff.ac.uk, aasaljohani@kau.edu.sa, Cardiff University, Cardiff, Wales, UK and King Abdulaziz University, Jeddah, Saudi Arabia; Hakan Kayan, kayanh@cardiff.ac.uk, Cardiff University, Cardiff, Wales, UK; Omer Rana, ranaof@cardif.ac.uk, Cardiff University, Cardiff, Wales, UK; Charith Perera, PereraC@cardif.ac.uk, Cardiff University, Cardiff, Wales, UK.

for TinyOL, the only one of the three able to operate without post-deployment labels. Like these approaches, we start from a backbone model pre-trained on labelled, anomaly-free data at design time, and then extend it to operate post-deployment, where labels are no longer accessible.

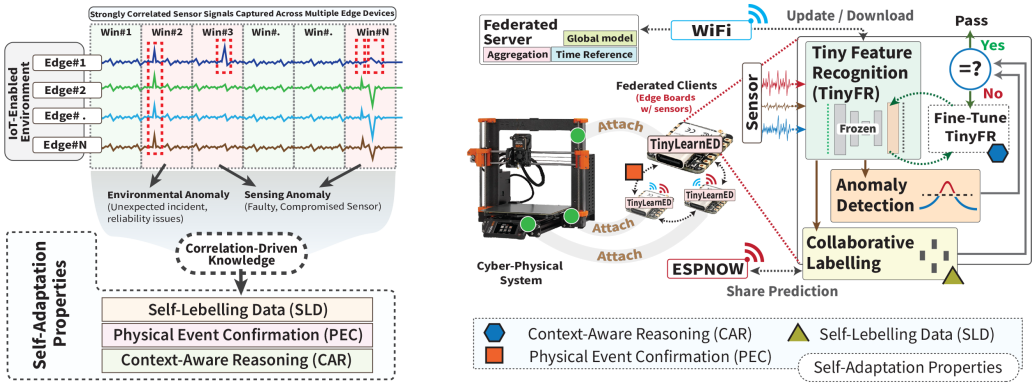
In this paper, we introduce **TinyLearnED**, an online, weakly supervised end-to-end anomaly detection (AD) framework for resource-constrained IoT devices. It uses correlations between devices monitoring the same physical environment to generate pseudo-labels, enabling models to adapt without ground-truth labels. To handle the inherent noise and potential adversarial nature of pseudo-labels, we incorporate a lightweight federated learning (FL) step that averages small final-layer parameter updates across devices to align their models and smooth update noise; the current design does not perform explicit outlier rejection. TinyLearnED has three main components: **Tiny Feature Recognition (TinyFR)**, which starts from a pre-trained AD model trained offline under normal conditions. TinyFR follows the same design principles as previous works [9, 28]. Therefore, we freeze all layers except the final layer on-device, which remains trainable. This approach preserves the initial representation while enabling rapid updates. **Online Fine-Tuning (OFT)** updates the model on-device using pseudo-labels from peer devices. **Label Noise Smoothing (LNS)**, which aggregates updates across devices to improve resilience. We evaluate TinyLearnED through chronological replay of physically collected Bolt 3D streams on four XIAO ESP32S3 side-channel monitoring (SCM) nodes. The data come from one Prusa MK4 printer producing a single bolt geometry with triaxial accelerometer sensing, under the tested normal, attack, and node-compromise conditions. TinyLearnED performs local inference and last-layer adaptation on the monitoring boards, with periodic federated parameter exchange, without requiring post-deployment ground-truth labels or full-model retraining during the evaluated streams. Thus, the system does not replace the complete deployed model over the air, although it does wirelessly exchange final-layer parameters. The results establish feasibility and task-dependent benefits in this setting, rather than cross-geometry transfer or long-term deployment stability. This paper makes three key contributions:

- **An online, weakly supervised AD framework for constrained IoT devices.** TinyLearnED supports on-device adaptation during post-deployment data streams without requiring ground-truth labels.
- **Bounded resilience to noisy or adversarial peers via lightweight federated aggregation.** We periodically average last-layer updates, providing implicit regularisation and improving performance under the evaluated noisy and compromised-peer conditions without requiring server-side data access.
- **Bounded real-world validation on a CPS testbed.** We evaluate TinyLearnED on four XIAO ESP32S3 SCM nodes using chronological replay of the Bolt 3D dataset, collected from one Prusa MK4 printer producing a single bolt geometry with triaxial accelerometer sensing. The experiments show task-dependent improvements over static and local-online-learning baselines under the tested attack and compromise configurations.

The remainder of the paper presents the motivation (Section 2), background (Section 3), use case and threat model (Section 4), the TinyLearnED design (Section 5), experimental setup (Section 6), evaluation (Section 7), discussion (Section 8), and conclusion (Section 9).

2 Motivation

Pre-trained TinyML models deployed at the edge become outdated as operating context drifts, and retraining in the cloud is expensive and often impractical. These challenges are acute in **Additive Manufacturing (AM)**: monitoring a 3D printer with TinyML models requires an adaptation mechanism that can respond to changes in the monitored signals, especially when sensors or the monitoring framework itself may be compromised during printing. To explore this, we equipped a 3D printer with multiple sensors and studied the correlation between sensor streams observing the



(a) Multi-Sensor correlation knowledge for SCM (b) TinyLearnED overview

Fig. 1. Motivation for TinyLearnED: multi-sensor correlation knowledge and the system overview.

same phenomena across repeated bolt-printing jobs under normal and controlled attack conditions (Figure 1(b)). We observed strong correlations between signals that monitor the same events, such as filament extrusion, bed movement, nozzle temperature changes, and anomalous behaviour. These correlations indicate that multiple edge boards observe complementary aspects of the same physical process. Such correlation-driven collaboration among edge boards enables the self-adaptation properties critical to resilient IoT systems [3] (Figure 1(a)). Agreement among correlated peers can supply pseudo-labels for streaming sensor data. These pseudo-labels form the basis for on-device fine-tuning of TinyML models on resource-constrained edge boards. Section 4 details the use case in which we assess this proposal.

3 Background and Related Work

Machine learning (ML) can now be deployed on resource-constrained devices at the edge due to the advent of TinyML, which compresses models via quantisation and pruning for microcontroller unit (MCU) hardware [30, 34]. Although it is common to perform inference on resource-constrained devices, training or adapting models locally remains difficult due to their limited memory and computational resources [19, 26]. Recent research has explored various approaches to enable on-device learning through model optimisation techniques and online continual learning frameworks.

Model optimisation techniques focus on making neural network training feasible within MCU constraints through algorithmic innovations. Checkmate [13] introduces dynamic tensor rematerialisation to address the memory bottleneck, where instead of storing an intermediate activation, it can be re-computed. Thus, this ultimately trades off computation for memory. POET [25] extends this with memory paging strategies for moving data between fast static random-access memory (SRAM) and slower flash storage. Nonetheless, the majority of these optimisation approaches aim to make training efficient rather than answering how a model should adapt to changing data or concepts once deployed. In other words, these works focus on efficient training, often under assumed static or fully supervised conditions, and leave open the problem of when and how to update the model in an evolving environment.

Online and continual learning frameworks enable models to incrementally update on streaming data without requiring batch retraining or cloud connectivity. TinyOL [28] implements unsupervised online learning on an Arduino Nano, incrementally fine-tuning a pre-trained autoencoder on new data based on reconstruction error. The architecture of LifeLearner [16] addresses the catastrophic forgetting issue of online machine learning (OML) by freezing most parts of a pre-trained model as a fixed feature extractor. Then it updates only the final classification layers. It also employs

Table 1. TinyLearnED in relation to foundational concepts in the literature.

Concept	Inherited Principle	Standard Assumption	TinyLearnED Adaptation for IoT/CPS
Co-training / tri-training [7, 39]	Agreement among multiple views or models assigns pseudo-labels to unlabeled data	Views are conditionally independent; labelling is typically performed centrally	Physical redundancy among spatially distributed SCM nodes observing the same CPS event provides an alternative basis for peer agreement
K-of-N ensemble voting	Aggregating binary decisions across nodes reduces single-node error	Classifiers or sensors are assumed to be mostly benign	K is configured based on fault-tolerance requirements, expected node compromise rate, and the recall/precision trade-off
FedAvg [21]	Averaging local model parameters across clients produces a shared global model	A central server controls training rounds; clients hold independent data batches	Only final-layer parameters are exchanged; aggregation acts as a low-cost stabiliser for noisy pseudo-label updates
TinyOL last-layer adaptation [28]	Restricting updates to the final layer enables online learning on MCUs	Incoming samples are assumed safe for direct online adaptation	Updates occur only on disagreement between the local prediction and the peer-consensus pseudo-label, which supplies the training target
Weak supervision [38]	Noisy or heuristic labels provide a useful training signal	Label noise is often treated as stochastic or modelled statistically	Noise may be adversarial, arising from compromised or manipulated SCM nodes

latent replay, which stores intermediate feature representations rather than raw data. The ML-MCU [35] proposes variants of stochastic gradient descent (SGD) optimised for binary and multi-class classification on MCUs. They achieve complete training loops on devices with limited resources. Ren et al. [29] have employed streaming gradient descent with complex event processing running on Cortex-M4 devices to enable real-time learning from sensor streams. All the works mentioned above demonstrate the technical feasibility of on-device adaptation for TinyML systems.

Nevertheless, despite all these improvements, there is a common limitation for existing approaches. Most prior efforts assume the presence of strong supervision during adaptation, requiring either labelled data or pre-defined triggers to guide model updates [9, 16]. TinyOL is fine-tuning on incoming sensing data. This creates a vulnerability in which transient anomalies or adversarial inputs can be maliciously learned as new normals, as TinyOL cannot label them to distinguish legitimate drift from transient noise. LifeLearner and similar frameworks require a small labelled dataset with explicit task boundaries to facilitate the classifier update without introducing false class labels [9, 16]. In practical IoT deployments, this supervision assumption rarely holds. Sensor data suffers from concept drift and anomalies, lacking a mechanism to provide real-time labels indicating whether observations are part of normal evolution or malicious activity. Our work addresses the aforementioned challenges by enabling collaborative adaptation across multiple edge devices sharing the same CPS context. TinyLearnED does not introduce a new general co-training algorithm or federated optimization algorithm; rather, it instantiates these principles under the constraints of deployed CPS monitoring. Specifically, it exploits the natural physical redundancy that arises when multiple spatially distributed side-channel monitoring (SCM) nodes observe the same physical process, as discussed in Section 2. In this setting, peer agreement serves not merely as a generic semi-supervised learning heuristic, but as a physical consistency check among independent monitors observing the same CPS event. At the adaptation level, TinyLearnED adopts the last-layer fine-tuning approach from TinyOL [28], restricting online updates to a small set of trainable parameters to respect MCU memory and latency constraints. This is extended through a coordinated pseudo-labelling and federated update strategy. Inspired by co-training and tri-training [7, 39], each edge node leverages agreement among peer devices to assign weak labels to new data points in real time. Local adaptation is triggered when a device's prediction diverges from the group consensus, which is then used as a weak supervisory signal [6, 27]. This reduces reliance on any

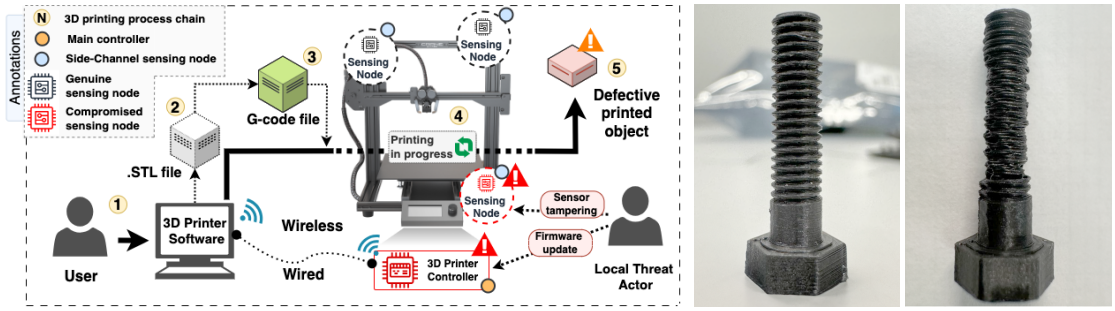
single device, whose predictions may be corrupted by anomalies, adversarial manipulation, environmental shifts, or sensor faults. To further mitigate noisy pseudo-label updates, only the unfrozen final-layer parameters are exchanged and aggregated via lightweight federated learning [3, 21]. In this role, FedAvg acts not as a mechanism to train a full global model from distributed data, but as a low-cost stabilizer that smooths weak last-layer updates generated from peer-consensus pseudo-labels. The key systems contribution of TinyLearnED is therefore the integration of weak peer-consensus labelling, selective last-layer online adaptation, and lightweight aggregation under a threat model where both the physical process and a subset of sensing nodes may be compromised. In contrast to approaches that rely on external labels or centralized retraining, TinyLearnED operates without external supervision while supporting adaptive and resilient post-deployment learning in resource-constrained IoT/CPS environments [38]. Table 1 summarises how TinyLearnED builds upon these foundational concepts from the literature while adapting their assumptions and mechanisms to the constraints of deployed IoT/CPS monitoring.

4 Cyber-Physical System Use Case

The process of AM, commonly referred to as 3D printing, builds three-dimensional objects through incremental material layering, offering design flexibility and resource efficiency unlike traditional manufacturing. Recent advances have driven AM growth across safety-critical and consumer applications in the biomedical, architectural, aerospace, and automotive sectors. However, widespread 3D printing adoption creates valuable targets for cyberattacks [5]. Compared to other CPSs, 3D printing involves interconnected processes from design to final object, with vulnerabilities throughout this chain. In [12], firmware attacks represent serious threats, allowing attackers to alter printing instructions, potentially modifying fan speed, layer thickness, nozzle movement, or infill patterns while bypassing inspections. Defence requires SCM using sensors separate from the printer, as examined by [12] using multiple sensing fusion models and [4] with a multilayered approach using microphones, inertial measurement unit (IMU) sensors, and tomography scanning. However, SCM techniques remain vulnerable as edge boards can be compromised through side-channel attacks [23], bypassing read-out protections [22], and firmware manipulation [31], ultimately compromising sensor data. An attacker with physical access can simultaneously target both printer firmware and SCM sensors. Previously mentioned approaches use a single controller that collects data from all sensors, creating a single point of failure. Distributing sensors across different controllers may mitigate this threat, as implemented by [6] to detect masking and spoofing attacks on smart home sensors using sensor fusion despite compromised sensors. This paper implements an SCM approach in which several edge boards learn collaboratively to identify 3D-printing anomalies and compromised sensors, demonstrated on domestic 3D printers as an accessible alternative to industrial-grade machines.

4.1 Threat Model

Our threat model considers two attack vectors: malicious firmware manipulation of 3D printers and spoofing attacks on independent SCM sensing nodes, as shown in Figure 2(a). We assume an adversary with access to digital inputs (STL/G-code files) and the printer's physical controller, enabling firmware modification and SCM compromise. Such access allows the attacker to alter printing parameters while bypassing internal checks and to manipulate external SCM sensors to conceal these changes. We illustrate this threat model using a 3D bolt object (Figure 2(b)). The key assumption is that the adversary can access both the printer and the SCM system, enabling coordinated cyber–physical attacks. Although potential attack techniques are outlined in Section 4, executing real attacks against firmware or SCM nodes is beyond the scope of this work. Based on this model, we categorise threats into two primary attack types:



(a) 3D printing use case (b) Normal (c) Defective

Fig. 2. Overview of a 3D printing system security model showing (a) the process chain and potential attack vectors, (b) a normally printed bolt object, and (c) a defective bolt resulting from tampering of printing setting.

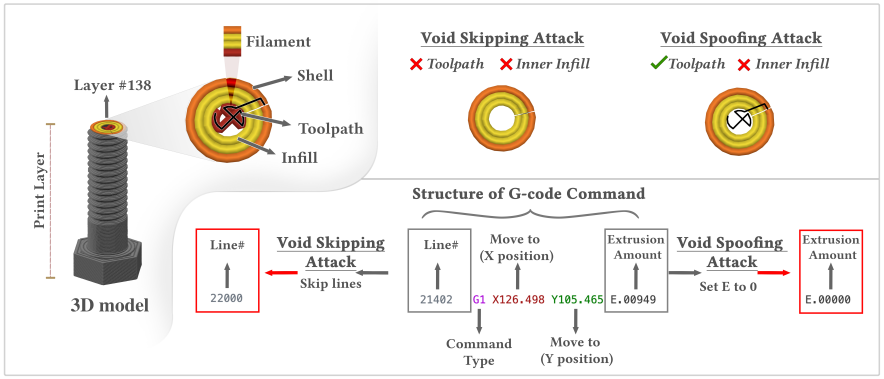


Fig. 3. Void attacks via G-code manipulation: Void Skipping (VS) removes toolpath/infill commands, while Void Spoofing (VSP) zeroes extrusion (E=0.00000) so the nozzle moves without depositing filament. Both create internal voids without visible surface defects.

FWC attacks. Firmware compromise enables unauthorised modifications of G-code instructions or printing parameters, producing defective objects by bypassing application layer protocols and manipulating low-level controller logic. We outline three scenarios: **1) *Layer Thickness (LT) attack*** manipulates layer height to degrade surface texture and precision. An attacker modifies the G-code to increase the layer height from 0.1 mm to 0.2 mm. This compromise affects aesthetic and functional properties by reducing surface smoothness and dimensional accuracy, as shown in Figure 2(c). ***2) Void Skipping (VS) attack*** targets internal structure by omitting infill deposition commands. An attacker may skip Line #21402 to #22000 in the G-code, bypassing infill commands across selected layers, as illustrated in Figure 3. Although the exterior remains intact, the internal gaps weaken the structural integrity, without visible external defects. ***3) Void Spoofing (VSP) attack*** manipulates extrusion values while maintaining nozzle trajectories. By setting the extrusion parameters to zero (E=0.00000), the nozzle moves along its path without releasing the filament, as demonstrated in Figure 3. This creates void areas while preserving the external appearance, making detection difficult through visual inspection or movement tracking alone.

SCMC attacks. SCM node compromise targets the SCM system to hide printing defects by suppressing or manipulating sensor readings that would otherwise detect anomalies. An attacker may coordinate a dual-layer attack, simultaneously

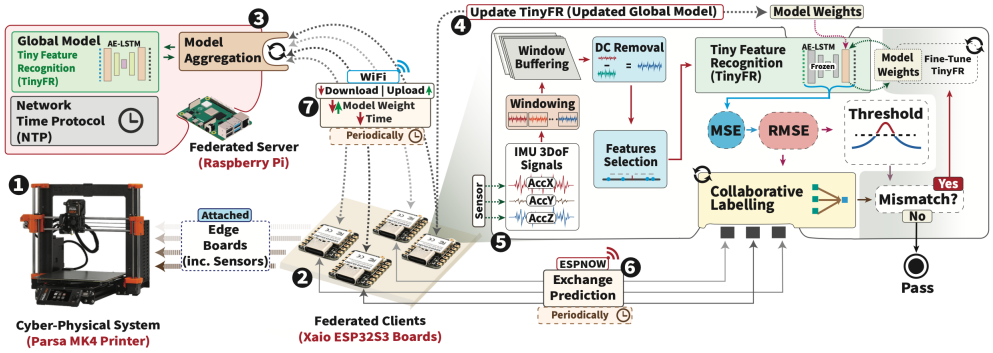


Fig. 4. TinyLearnED System Architecture

compromising the 3D printer firmware and the SCM system. This approach can bypass internal software validation or inspection by the printer controller while evading external SCM detection, enabling a strategic coordinated attack. We apply TinyLearnED as a collaborative sensing framework enabling learning between SCM nodes. Our objective is to validate the behaviour of the SCM sensing node during the printing task in the presence of a compromised node.

5 Approach

5.1 System Design

In this section, we describe the design of the TinyLearnED system and its integration with the CPS, as illustrated in Figure 4. The architecture includes multiple edge boards (e.g., XIAO ESP32S3) attached to the CPS (i.e., Prusa MK4 3D printer). These boards serve as sensor nodes in SCM. We selected these edge boards based on a correlation analysis of their sensor readings. This approach ensures that edge boards observing similar phenomena participate in collaborative learning, see Section 6.1.4. Each board can run a TinyML model, which enables on-board inference for real-time AD. TinyLearnED is installed on each edge board to enable boot-up and establish wireless communications. We use two communication protocols to facilitate interaction between each edge board and its corresponding peers, along with the FL server. For peer-to-peer data exchange between edge boards, we adopt Espressif ESP-NOW (ESPNOW) [11]. Our experimental stack uses ESP-NOW with a maximum application payload of 250 bytes [11]. This setup enables efficient sharing of anomaly scores or predictions among edge boards. In contrast, larger payloads (e.g., model weights and biases) require an intermittent WiFi connection to communicate with a central FL server. During initialisation, each edge board uploads the weights and biases of the unfrozen layers to the FL server. The server performs aggregation to create the initial global model (see Section 5.7). The FL server also runs a Network Time Protocol (NTP) service for timestamp synchronisation across clients. Following initialisation, the edge boards capture IMU signals from the CPS. These raw signals go through several processing steps to retain meaningful temporal patterns (see Section 5.4). The processed features are fed into the **Tiny Feature Recognition (TinyFR)** model. This model uses a compressed autoencoder (AE)–Long Short-Term Memory (LSTM), or AE-LSTM, architecture optimised for edge deployment. TinyFR calculates the **Mean Squared Error (MSE)** and **Root Mean Squared Error (RMSE)** between the input and its reconstruction. If the RMSE exceeds a predefined threshold, the processed windows are identified as anomalous (see Section 5.5). Each edge board makes local binary decisions and transmits these decisions via ESPNOW. These edge boards, in parallel, wait for binary decisions from their peers. Once the exchange is complete, each board obtains a pseudo-label through collaborative labelling (Section 5.6). The board initiates **Online Fine Tuning (OFT)** of TinyFR's final trainable layer if

and only if its local binary prediction disagrees with this pseudo-label, as defined in Equation (4). The pseudo-label provides the training target and may be either normal (0) or anomalous (1). Agreement between the local prediction and the pseudo-label skips the local OFT update (Section 5.8).

At regular intervals, the FL server receives updates on the weights and biases of the unfrozen layers from each edge board. This process enables the FL global model to evolve over time and enhances generalisation across devices operating under slightly varying conditions.

5.2 Modelling TinyLearnED-Enabled CPSs

To formally explain the behaviour of `TinyLearnED` under both normal and adversarial conditions, we describe the system during normal operation and then clarify which components are impacted by the threat model in Section 4.1.

System modelling. We model the 3D-printing use case as a CPS = (E, A, M), where E denotes the physical environment (nozzle movements, extruder temperature, fan state, material flow rate), $A = \{a_j \mid j = 0, \ldots, N-1\}$ is the set of N SCM edge boards with a_{N-1} the designated FL server, and M the set of exchanged messages. Each board a_j observes the environment through its IMU, producing the three-axis acceleration signals $E_j = \{\texttt{accX}^{(j)}, \texttt{accY}^{(j)}, \texttt{accZ}^{(j)}\}$. During a 3D printing task, sensor values are collected over sliding windows ω_t of length L with stride S. At each time step, the IMU records three-axis acceleration measurements. A window is defined as:

$$\omega_t^{(j)} = \left\{ \left[p_{0,t+k}^{(j)},\ p_{1,t+k}^{(j)},\ p_{2,t+k}^{(j)} \right] \,\middle|\, k = 0, \ldots, L-1 \right\} \in \mathbb{R}^{L \times 3}. \tag{1}$$

For continuous monitoring, windows are generated as $\omega_t^{(j)},\ \omega_{t+S}^{(j)},\ \omega_{t+2S}^{(j)},\ \ldots$. Each edge board a_j then derives features from $\omega_t^{(j)}$ for anomaly detection and shares only window-level anomaly decisions with its peers. To support collaborative detection and adaptation, each board participates in decision-level message exchange. The prediction-sharing channel is:

$$m_{a_j}^{\text{pred}}(t) \in \{0, 1\} \rightarrow \{a_k \mid k \neq j\}. \tag{2}$$

Raw windows $\omega_t^{(j)}$ are not transmitted to peers. Each board aggregates peer decisions using an ensemble function $\mathcal{F}_{\text{ens}}$ to generate a pseudo-label:

$$\tilde{y}_j(t) = \mathcal{F}_{\text{ens}} \left(\left\{ m_{a_k}^{\text{pred}}(t) \mid k \neq j \right\} \cup \left\{ m_{a_j}^{\text{pred}}(t) \right\} \right). \tag{3}$$

Each board compares its local binary prediction with the pseudo-label for the same window. Local OFT is triggered if and only if

$$m_{a_j}^{\text{pred}}(t) \neq \tilde{y}_j(t). \tag{4}$$

Let $\theta_j = (\mathbf{W}_j, \mathbf{b}_j)$ denote the final-layer parameters on board a_j, and let $\mathbf{G}_j(t) = (\mathbf{G}_{W,j}(t), \mathbf{G}_{b,j}(t))$ denote the BCE-derived surrogate update signals computed by Algorithm 1. The local OFT update is

$$\theta_j \leftarrow \begin{cases} \text{Update}\big(\theta_j, \mathbf{G}_j(t)\big), & m_{a_j}^{\text{pred}}(t) \neq \tilde{y}_j(t), \\ \theta_j, & m_{a_j}^{\text{pred}}(t) = \tilde{y}_j(t). \end{cases} \tag{5}$$

Here, Update applies Adam with persistent optimiser state. The pseudo-label supplies the training target, whereas disagreement determines whether the local update is executed. Finally, each client board $\{a_j \mid j = 0, \ldots, N-2\}$ periodically transmits its updated parameters $\theta_j^{(t)}$ to the FL server a_{N-1}, which aggregates them to produce a global model:

$$\bar{\theta}^{(t)} = \text{Aggregate} \left(\left\{ \theta_j^{(t)} \mid j = 0, \ldots, N-2 \right\} \right). \tag{6}$$

The FL server distributes $\bar{\theta}^{(t)}$ to the participating edge boards, improving generalisation across devices.

Adversarial modelling As described in Section 4.1, we extend the CPS model to an adversarial form to capture compromise scenarios:

$$CPS' = (E', A', M', C). \tag{7}$$

Here, C denotes the set of adversarial actions. A 3D printer firmware compromise (FWC) alters the physical behaviour of the CPS and therefore affects only the environment component E'. In contrast, an SCM node compromise (SCMC) affects E', A', and M', because the adversary gains full access to a compromised board $a'_j \in A'$ (i.e., corresponding to some benign board $a_j \in A$) and can manipulate both local sensing and transmitted messages. In particular, the adversary can overwrite the reported anomaly decision to bypass SCM detection, forcing a compromised node to always report *normal*, $m^{\text{pred}}_{a'_j}(t) := 0$, regardless of its honest local decision, or conversely to always report *anomalous*, $m^{\text{pred}}_{a'_j}(t) := 1$. Forced-normal manipulation suppresses detection of genuine defects, whereas forced-anomaly manipulation injects spurious votes that corrupt collaborative pseudo-labelling and trigger unnecessary OFT updates; Section 5.10 analyses the consensus rule's tolerance to both.

5.3 Edge Board Selection

Edge board selection precedes the implementation of TinyLearnED, with correlation-based knowledge illustrated in Figure 1(a). Boards are evaluated based on their ability to acquire contextual information about the underlying CPS, as physical events generate unique signatures that affect edge board responses differently. Since sensor signals may precede, follow, or overlap events, causing variable detection times, determining the optimal window size across all edge boards becomes critical. Therefore, a grid search is used to identify the optimal window length for edge board alignment in printing tasks. The selection process involves outlier removal to exclude sensor data anomalies unrelated to CPS operation using rolling median filters with small window lengths, thus balancing outlier elimination and information preservation. Furthermore, edge board groups influenced by identical physical events are identified by rolling statistical feature computation from sensor signal windows, with the **Pearson Correlation Coefficient (PCC)** applied to assess linear correlation between event ground truth and sensor signals, as detailed in Section 6.1.4.

5.4 Data processing

Data processing in TinyLearnED begins with the construction of sliding windows from raw IMU signals (AccX, AccY, AccZ) collected by edge boards during 3D printer operation. These windows are buffered for sequential processing. The process starts with direct-current (DC) removal applied to each window to eliminate constant offsets in the signal that do not reflect the dynamics of the printer, centring the signal, and enhancing the sensitivity of downstream processes to transient events. The next step involves feature selection on each denoised and centred window, extracting statistical features that summarise the vibrational patterns within each window.

5.5 TinyFR Model

The complete workflow for training and deploying the TinyFR model on edge boards is illustrated in Figure 5. We design TinyFR using a sequence-to-sequence AE-LSTM architecture that learns the *normal* temporal patterns of 3D printer operation. In Section 6.2, we explain the rationale for choosing AE-LSTM.

The input window consists of T=3 time steps, each containing F=9 statistical features. These features are extracted during the data processing stage, as detailed in Section 5.4. We denote the input window as $\mathbf{P} \in \mathbb{R}^{T \times F}$. The model is trained to reconstruct the input by minimising the mean squared error (MSE) between $\mathbf{P}$ and its reconstruction $\hat{\mathbf{P}}$:

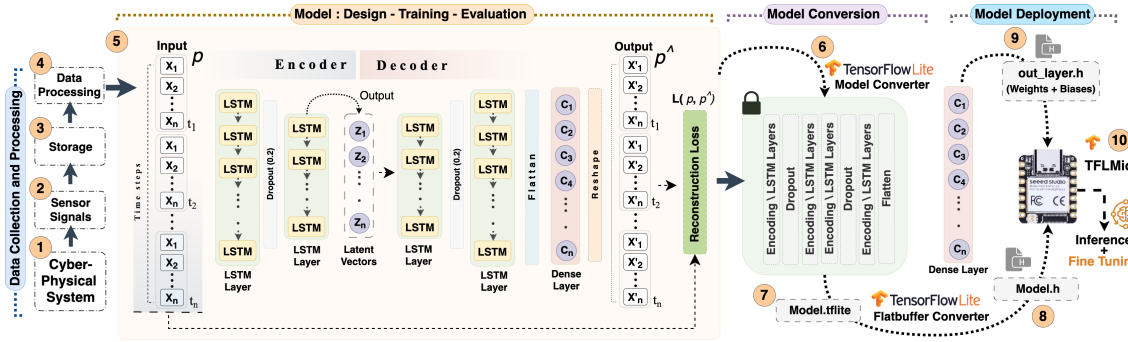


Fig. 5. Workflow for training and exporting TinyFR

$$\mathcal{L}_t(\mathbf{P}, \hat{\mathbf{P}}) = \frac{1}{F} \sum_{f=1}^{F} \left(P_{t,f} - \hat{P}_{t,f}\right)^2, \quad \text{for } t = 1, \ldots, T. \tag{8}$$

Network architecture The network follows a standard encoder–decoder architecture based on LSTM layers. The encoder processes the input sequence $\mathbf{P}$ through two stacked LSTM layers. The first layer contains 16 units and outputs hidden states $\mathbf{H}_1 = \text{LSTM}_{16}(\mathbf{P})$, where $\mathbf{H}_1 \in \mathbb{R}^{3\times16}$. The second layer is an LSTM with 4 units; its output is the latent representation $\mathbf{Z} = \text{LSTM}_4(\mathbf{H}_1)$, where $\mathbf{Z} \in \mathbb{R}^{3\times4}$. To avoid overfitting during training, a dropout layer with probability $p = 0.2$ follows each hidden state.

The decoder passes the latent sequence $\mathbf{Z}$ through two LSTM layers with 4 and 16 units, respectively. The first decoder layer outputs intermediate states $\tilde{\mathbf{H}}_1 = \text{LSTM}_4(\mathbf{Z})$, where $\tilde{\mathbf{H}}_1 \in \mathbb{R}^{3\times4}$, and the second outputs $\tilde{\mathbf{H}}_2 = \text{LSTM}_{16}(\tilde{\mathbf{H}}_1)$, where $\tilde{\mathbf{H}}_2 \in \mathbb{R}^{3\times16}$. We then flatten this output as $\tilde{\mathbf{H}}_3 = \text{flatten}(\tilde{\mathbf{H}}_2)$, yielding $\tilde{\mathbf{H}}_3 \in \mathbb{R}^{1\times(3\times16)}$, and pass it to a dense layer $\tilde{\mathbf{H}}_4 = \text{Dense}(\tilde{\mathbf{H}}_3)$ with $\tilde{\mathbf{H}}_4 \in \mathbb{R}^{1\times(3\times9)}$ neurons. The dense output is reshaped into the reconstructed input $\hat{\mathbf{P}} \in \mathbb{R}^{T\times F}$ to match the original input $\mathbf{P}$. The model is trained using SGD with a learning rate of $\eta = 10^{-3}$ until the reconstruction loss converges.

Anomaly Detection Following training, we utilise a threshold-based AD method. After each inference instance, we calculate the RMSE and compare it against a threshold. Since $\mathcal{L}_t(\mathbf{P}, \hat{\mathbf{P}})$ is defined in Eq. 8, the RMSE for each time step can be calculated as in Eq. 9.

$$RMSE_t = \sqrt{\mathcal{L}_t(\mathbf{P}, \hat{\mathbf{P}})}, \quad \text{for } t = 1, \ldots, T \tag{9}$$

Model conversion Because edge boards have limited resources, retraining the full model on-board is impractical. Therefore, after offline training we compress the TinyFR layers and maintain only the final dense output layer as trainable, similar to [28]. The trainable layer consists of a weight matrix $\mathbf{W} \in \mathbb{R}^{27\times48}$ and a bias vector $\mathbf{b} \in \mathbb{R}^{27}$. The final reconstruction is computed online on the edge board as:

$$\hat{\mathbf{P}} = \text{reshape}\left(\mathbf{W}\ \text{flatten}(\tilde{\mathbf{H}}_2) + \mathbf{b}\right). \tag{10}$$

To deploy TinyFR on edge boards, we start by converting and optimising the encoder-decoder layers using the TFLite converter. This process produces a `model.tflite` file with post-training quantisation to **16-bit Floating Point (float16)**. Next, we use the Flatbuffer converter tool provided by TFLite to transform the `model.tflite` file into a C array stored in a header file (i.e., `model.h`). The TFLite Micro runtime then uses this C array to perform inference on edge boards. Finally, the weights $\mathbf{W}$ and the bias vector $\mathbf{b}$ of the last dense layer are inserted into a separate header file (`out_layer.h`).

This file includes other pre-defined variables. For instance, it contains the best threshold for AD determined via grid search, as well as the mean and variance from the training set used for standardising the features on the edge boards when evaluated on the testing set. These files can then be compiled to edge boards to run inference and facilitate OFT without requiring a recompilation of the firmware.

5.6 Collaborative Labelling

Obtaining reliable labels for on-device adaptation is often expensive or impractical. Following the co-training principle [7, 18], TinyLearnED uses predictions from physically co-located edge boards observing correlated CPS events to generate pseudo-labels. Each board processes its local sensor stream and exchanges a binary prediction with its peers for the same window. The N predictions, including the local board's prediction, are combined using an unweighted K-of-N rule: the pseudo-label is 1 if at least K boards report an anomaly, and 0 otherwise. This voting mechanism provides a straightforward basis for collaborative decisions [6, 27].

Pseudo-label generation and OFT activation are separate steps. After obtaining the pseudo-label, each board compares it with its own local prediction. The pseudo-label is used as the training target only when these decisions disagree, as specified in Equation (4); agreement produces no local OFT update. Thus, the rule does not require all boards to report 1, and an anomalous pseudo-label alone does not trigger OFT.

For a fixed set of predictions, lowering K assigns more windows an anomalous pseudo-label, increasing sensitivity but potentially admitting more false alarms. Raising K requires broader support for an anomaly label but may miss events observable at fewer sensor placements. The number of OFT updates depends on disagreements with these pseudo-labels, not simply on the number of anomaly votes. In our deployment with $N = 4$ boards, we use $K = 3$; the rationale and robustness limits are analysed in Section 5.10.

5.7 Federated Aggregation to Stabilise Pseudo-Label OFT

TinyLearnED adopts a bottom-up federated learning (FL) strategy. Unlike conventional FL, where a server initialises and distributes a global model, our server is initialised without a global model. Each edge board uploads the parameters of TinyFR's unfrozen (trainable) layer to the server, which aggregates them via simple averaging (FedAvg) to bootstrap an initial global layer [21]. The aggregated layer is then broadcast to all participating boards. During deployment, each board performs local OFT only when its prediction disagrees with the peer-consensus pseudo-label, and it periodically uploads its current final-layer parameters to the server. The federated aggregation interval is independent of the disagreement-based OFT trigger. Aggregation acts as a lightweight regulariser by averaging updates across boards. The resulting global layer is less sensitive to device-specific drift and to variance introduced by noisy pseudo-labels, which improves consistency across edge devices while preserving data locality.

5.8 Online Fine Tuning

Our OFT approach extends TinyML inference by selectively updating the final trainable layer of TinyFR one window at a time. The model first performs inference and records its local binary prediction before any update on that window. After collaborative labelling, OFT is executed if and only if this prediction disagrees with the peer-consensus pseudo-label (Equation (4)). Both disagreement cases trigger an update: a local normal prediction with an anomalous pseudo-label, and a local anomalous prediction with a normal pseudo-label. The pseudo-label supplies the training target in either case. When the two decisions agree, the local optimiser step is skipped, including when both decisions are 1. This selective update rule avoids an OFT computation on agreeing windows without restricting training to anomaly pseudo-labels.

Algorithm 1 Online Fine-Tuning (OFT) of the Trainable Fully-Connected Layer in TinyLearnED

Require: Latent vector $\mathbf{z} \in \mathbb{R}^{48}$; target window $\mathbf{P} \in \mathbb{R}^{T\times 9}$, $T = 3$; pseudo-label $\tilde{y} \in \{0, 1\}$; threshold τ; learning rate η; sensitivity α; $\epsilon = 10^{-15}$

Require: Invoke PartialFitBCE if and only if $m_{a_j}^{\text{pred}}(t) \neq \tilde{y}_j(t)$, with $\tilde{y} = \tilde{y}_j(t)$; otherwise skip local OFT.

1: Load $\mathbf{W} \in \mathbb{R}^{27\times 48}$ and $\mathbf{b} \in \mathbb{R}^{27}$ from offline training ▷ Once
2: Initialise Adam moment buffers $\mathcal{A}$ to zero and $t_{\text{Adam}} \leftarrow 0$ ▷ Persistent state
3: **procedure** PartialFitBCE($\mathbf{z}, \mathbf{P}, \tau, \tilde{y}, \alpha$)
4: $t_{\text{Adam}} \leftarrow t_{\text{Adam}} + 1$
5: $\hat{\mathbf{p}}_{\text{flat}} \leftarrow \mathbf{W}\mathbf{z} + \mathbf{b}$ ▷ Flattened reconstruction
6: $\hat{\mathbf{P}} \leftarrow \text{reshape}(\hat{\mathbf{p}}_{\text{flat}}, (T, 9))$
7: $\text{MSE}_t \leftarrow \frac{1}{9}\sum_{f=1}^{9}(\hat{P}_{t,f} - P_{t,f})^2, \quad t = 1, \ldots, T$
8: $\text{RMSE} \leftarrow \frac{1}{T}\sum_{t=1}^{T}\sqrt{\text{MSE}_t}$ ▷ Mean per-timestep RMSE
9: $s \leftarrow \left(1 + \exp[-\alpha(\text{RMSE} - \tau)]\right)^{-1}$
10: $\mathcal{L} \leftarrow -\tilde{y}\log(s + \epsilon) - (1 - \tilde{y})\log(1 - s + \epsilon)$ ▷ Monitoring loss
11: $g \leftarrow \frac{s-\tilde{y}}{s(1-s)+\epsilon}\, s(1 - s)\alpha$ ▷ BCE-derived signal
12: $\mathbf{e} \leftarrow \hat{\mathbf{p}}_{\text{flat}} - \text{flatten}(\mathbf{P})$
13: $\mathbf{r} \leftarrow \mathbf{0}$ if RMSE $< 10^{-6}$; otherwise $\mathbf{r} \leftarrow \mathbf{e}/(T\,\text{RMSE})$
14: $\boldsymbol{\delta} \leftarrow g\mathbf{r}$ ▷ Surrogate output signal
15: $\mathbf{G}_W \leftarrow \boldsymbol{\delta}\mathbf{z}^\top$ ▷ Weight update signal
16: $\mathbf{G}_b \leftarrow \boldsymbol{\delta}$ ▷ Bias update signal
17: Update $\mathbf{W}, \mathbf{b}$ and $\mathcal{A}$ using Adam with $(\mathbf{G}_W, \mathbf{G}_b, t_{\text{Adam}}, \eta)$
18: **return** $\mathcal{L}$ ▷ Returned for monitoring
19: **end procedure**

TinyFR is trained offline (Section 5.5) and then adapted during the streaming evaluation without revisiting earlier windows. Ground-truth labels are not used to trigger or supervise these online updates.

Algorithm 1 details the update of the trainable dense layer, executed with batch size 1 [28]. Each update takes the target window $\mathbf{P} \in \mathbb{R}^{T\times 9}$ ($T = 3$), the flattened latent vector $\mathbf{z} \in \mathbb{R}^{48}$, the pseudo-label $\tilde{y} \in \{0, 1\}$, the threshold τ from offline training, the learning rate η for the weights $\mathbf{W} \in \mathbb{R}^{27\times 48}$ and bias $\mathbf{b} \in \mathbb{R}^{27}$, and the sensitivity α. The algorithm reconstructs $\hat{\mathbf{P}} = \text{reshape}(\mathbf{W}\mathbf{z} + \mathbf{b})$, computes the mean per-timestep RMSE against $\mathbf{P}$, and converts this raw error into a smooth confidence score

$$s = \frac{1}{1 + \exp\big(-\alpha\,(\text{RMSE} - \tau)\big)} \in (0, 1), \tag{11}$$

where α controls the sharpness of the sigmoid: errors below τ yield $s < 0.5$ and larger deviations $s > 0.5$, giving a differentiable estimate of anomaly likelihood. The implementation computes a binary cross-entropy value against the pseudo-label for monitoring:

$$\mathcal{L} = -\tilde{y}\log(s + \epsilon) - (1 - \tilde{y})\log(1 - s + \epsilon). \tag{12}$$

Here, $\epsilon = 10^{-15}$ is the numerical safeguard used in the loss and BCE-derived scalar signal. Algorithm 1, lines 11–16, combines this scalar signal with the reconstruction residual normalised by T RMSE, setting the normalised residual to zero when RMSE $< 10^{-6}$. This aggregate-normalised rule is a surrogate update, rather than the exact gradient of Equation (12) through the mean per-timestep RMSE. The resulting signals $\mathbf{G}_W$ and $\mathbf{G}_b$ are passed to Adam at learning rate η, with moment buffers and the step counter retained between calls. This is the update rule used in the reported TinyLearnED OFT experiments. The BCE value is returned for monitoring, and the window data are discarded after processing.

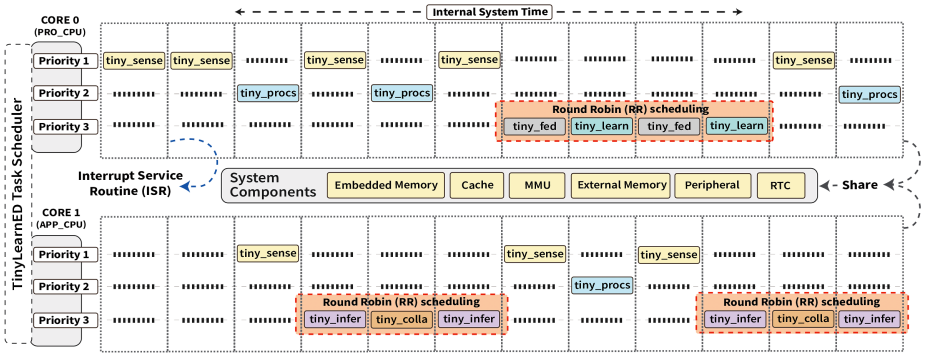


Fig. 6. TinyLearnED scheme task scheduling and prioritisation for Multi-core Processors

5.9 Task Scheduling and Prioritisation

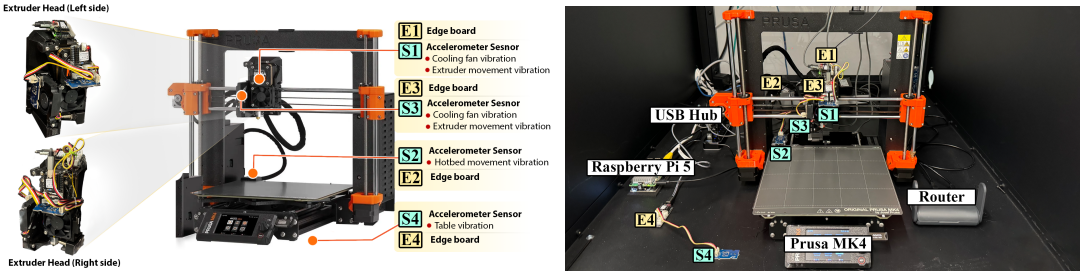
The ESP32-S3 MCU has two Xtensa LX7 cores. *Core 0 (PRO_CPU)* handles protocol-related activities, while *Core 1 (APP_CPU)* runs application-level code. TinyLearnED requires the concurrent execution of several tasks, including sensor data acquisition, preprocessing, TinyFR inference, collaborative labelling, participation in FL training rounds, and OFT. Therefore, we used FreeRTOS, a real-time operating system for the MCU, to provide task scheduling and manage internal components of the ESP32S3 boards. However, the default FreeRTOS scheduler for tasks with *no-affinity* can result in frequent inter-core task switching. This switching leads to cache failures and unpredictable latencies. Migration of tasks between cores disrupts cache locality and increases the variability of execution time [8, 32]. These effects are well documented in multicore real-time systems, where task migration can significantly degrade performance and latency [37]. To address these issues, we implemented a lightweight custom scheduler with four main features as shown in Figure 6: (1) Core-specific assignment ensures that time-critical tasks remain in their designated core. (2) Priority-based preemption allows higher-priority tasks (e.g., `tiny_infer`, `tiny_learn`) to interrupt lower priority tasks when necessary. (3) Non-affinity-based tasks employ the default *no-affinity* FreeRTOS scheduler. (4) Round-robin scheduling among tasks of equal priority offers fairness without compromising determinism. For complete tasks description and scheduling prioritisation, refer to the appendix A.

5.10 Design Guidelines and Robustness Analysis

TinyLearnED uses three coupled design parameters during online adaptation: the voting threshold (K), the number of participating edge boards (N), and the federated aggregation interval (R). These parameters control the trade-off between fast adaptation, pseudo-label quality, communication overhead, and robustness to compromised nodes. This subsection provides design guidelines for selecting these parameters and clarifies the expected robustness envelope of the system. We empirically evaluate these behaviours in Section 7.4.

5.10.1 Effect of the K-of-N Consensus Rule. The K-of-N rule controls the operating point of collaborative pseudo-labelling. Let p_d denote the probability that an honest edge board detects an anomalous window and p_f the probability that it raises a false alarm on a normal window. Under a simplifying assumption of conditionally independent board errors, the probability that the consensus fires on a window whose per-board trigger probability is p is

$$Q(p; K, N) = \sum_{i=K}^{N} \binom{N}{i} p^i (1-p)^{N-i}, \qquad P_D = Q(p_d; K, N), \quad P_{FA} = Q(p_f; K, N). \tag{13}$$



(a) (b)

Fig. 7. (a) Sensor placement (S1–S4) on the Prusa MK4 with dedicated XIAO ESP32S3 boards (E1–E4). (b) Full experimental setup.

These expressions provide design guidance rather than precise prediction, since sensors mounted on the same CPS may exhibit correlated errors, but they expose the central trade-off: a smaller K increases P_D and enables the system to respond to weak or localised anomalies, at the cost of a higher P_{FA}, which introduces noisy pseudo-labels into the OFT. Conversely, a larger K suppresses isolated false alarms but may fail to detect anomalies that are observable only from specific sensor placements. This trade-off is particularly significant under class imbalance. Given anomaly prevalence π, the expected precision of consensus-generated anomaly labels is given by

$$Precision(K, N) = \frac{\pi P_D}{\pi P_D + (1-\pi)P_{FA}}, \tag{14}$$

even a minor increase in P_{FA} reduces pseudo-label precision when anomalies are infrequent. Consequently, TinyLearnED applies a conservative threshold in adversarial deployments. For example, with $N = 4$ boards, setting $K = 3$ implements a strict-majority rule under which a single inconsistent board cannot determine the pseudo-label, while an anomaly is still accepted if one board fails to detect it.

5.10.2 Robustness Envelope under Compromised Nodes. The same rule defines tolerance to compromised SCM nodes. Let c boards be compromised. Under a *forced-normal* attack, compromised boards always report 0, so only $N - c$ honest boards can contribute anomaly votes and the detection probability falls to $P_D^{(c)} = Q(p_d; K, N - c)$; if $K > N - c$, the adversary blocks any anomaly consensus even when every honest board detects the event. Under a *forced-anomaly* attack, compromised boards contribute c unconditional anomaly votes, and if $K \leq c$ they can force an anomalous pseudo-label without support from any honest board. Tolerating up to $c_{\max}$ compromised boards against both behaviours therefore requires

$$c_{\max} < K \leq N - c_{\max}. \tag{15}$$

which jointly implies $2c_{\max} < N$: no choice of K guarantees correct consensus once the adversary controls half or more of the boards. This bound matches the empirical degradation in Table 6: with $N = 4$ and $K = 3$, TinyLearnED loses only 6–7 F1 points for LT and VS at $C1$, degrades sharply at $C2$ ($c = N/2$), and fails at $C3$, exactly as Equation (15) predicts. Because the binomial model assumes a homogeneous p_d while detection is in practice placement-dependent, the bound should be read with respect to the number of boards that can physically observe a given anomaly class rather than the nominal deployment size; this distinction is most visible for VSP (Section 8.3).

5.10.3 Aggregation Interval and Communication–Stability Trade-off. The aggregation interval R specifies the number of inference windows between federated exchanges. TinyLearnED exchanges only the final-layer weights and biases, comprising $P = 27 \times 48 + 27 = 1{,}323$ parameters. With FP32 serialisation ($q = 4$ bytes per parameter), the parameter payload is $B_{\text{param}} = Pq = 5{,}292$ bytes per upload or download, excluding protocol metadata and transport overhead. This payload exceeds the 250-byte application-payload maximum used in our stack for a single application packet [11], motivating the WiFi path for federated parameters while ESP-NOW carries the smaller peer-decision payloads.

Using the application-payload accounting in Table 7, the amortised communication per board and inference window is

$$B_{\text{app,window}} = B_{\text{peer,window}} + \frac{2B_{\text{param}}}{R}. \tag{16}$$

The peer-exchange contribution is fixed at 96 bytes per window in the reported four-board configuration. The federated component scales as $1/R$, giving total application payloads of approximately 308 and 167 bytes per window for $R = 50$ and $R = 150$, respectively. These quantities exclude transport overhead and retransmissions.

A smaller R shares local updates sooner, but increases communication and may also disseminate noisy updates more quickly. A larger R reduces the federated communication component while allowing local models to evolve for longer between exchanges; it does not reduce the fixed peer-exchange cost. The results in Section 7.3.3 therefore support a task-dependent trade-off rather than a universally preferable aggregation interval.

5.10.4 Practical Parameter Selection. The preceding analysis leads to three deployment guidelines. First, K should be selected according to the expected compromise tolerance. For a deployment that must tolerate up to $c_{\max}$ compromised boards, K should satisfy Equation 15. If no adversarial compromise is expected and recall is more important than precision, a lower K can be used to improve sensitivity. In adversarial CPS monitoring, however, a strict-majority rule is a safer default. Second, R should be selected from a cost-performance curve. Lower R increases communication and power consumption but reduces local model drift. Higher R reduces communication overhead but can allow noisy updates to persist for longer. Third, K and R should be considered jointly. For a fixed set of predictions, a lower K assigns more anomalous pseudo-labels, while a higher K requires broader support for an anomaly label. Neither setting directly specifies the OFT update frequency: updates occur only when a board's local prediction disagrees with its pseudo-label. Consequently, K affects which windows trigger local adaptation, whereas R determines how often the resulting parameters are aggregated. Their joint selection should account for pseudo-label quality, local disagreement, and communication cost; a lower K does not necessarily produce more OFT updates.

Table 2. Summary of 3D Bolt Dataset

Source	Features	#Data Points	Size
E1*	Acceleration (X, Y, Z)	18,812,007	737.7 MB
E2*	Acceleration (X, Y, Z)	18,811,959	738.9 MB
E3*	Acceleration (X, Y, Z)	18,812,079	745.6 MB
E4*	Acceleration (X, Y, Z)	18,811,902	739.6 MB
MK4 Log**	Extrusion Rate, Movement Speed, Printer State, Head Fan Speed, Z-Position, Nozzle Fan Speed	14,930,730	565.0 MB
Total	-	**90,178,677**	**3.53 GB**

** Collected via side-channel monitoring edge boards. ** Exported via Prusa MK4 internal API (Prusa Firmware Buddy).*

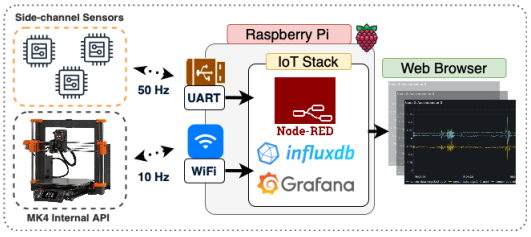
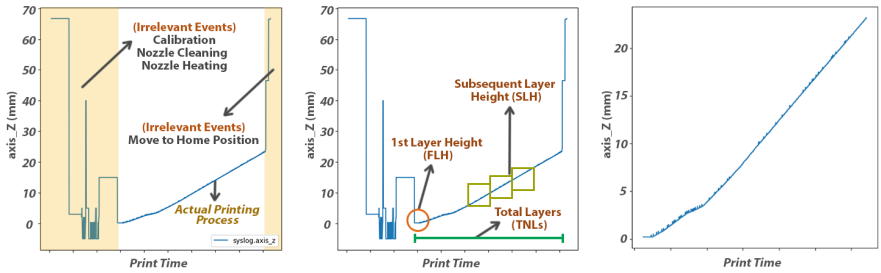


Fig. 8. The real-time data collection setup.



(a) (b) (c)

Fig. 9. Analysis of 3D printing process data using nozzle height (axis_z). (a) Raw axis_z data showing irrelevant events and the actual printing process. (b) Extraction of key printing parameters from axis_z: First Layer Height (FLH), Subsequent Layer Height (SLH), and Total Number of Layers (TNL). (c) Filtered axis_z data representing the actual printing process after removing irrelevant events.

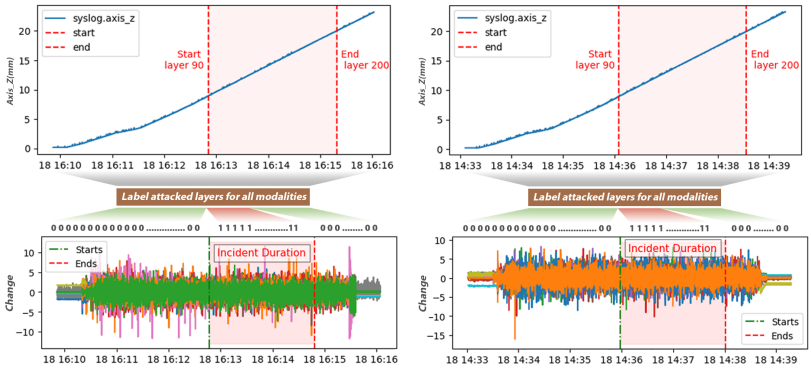
6 Experimental Setup

6.1 Dataset Creation

To evaluate the feasibility of TinyLearnED, we created the *Bolt 3D* dataset, which includes both anomalous and non-anomalous printing tasks. As shown in Figure 7(a), we placed multiple sensors and edge boards around a Prusa MK4 3D printer. Each sensor was connected to an XIAO ESP32S3 edge board via the I2C interface. This setup allows the edge boards to read sensor signals and transmit the data wirelessly (or via UART) to a Raspberry Pi, which serves as the data collector. As illustrated in Figure 8, the Raspberry Pi runs a complete IoT stack comprising InfluxDB, Grafana, and Node-RED. Each edge board operates as an independent SCM sensing node, isolated from the printer's internal network. This separation is achieved by configuring two distinct networks using a GL.iNet GL-AXT1800 router running OpenWrt. Sensor data were collected at 50 Hz, capturing the physical signatures generated by the MK4 during printing.

6.1.1 Data collection. We collected data from 57 printing jobs, divided into four print categories. First, we recorded sensing data for 30 non-anomalous prints, which were used to establish a baseline for non-anomalous printing operations on the MK4 printer. Each printing task lasted approximately 11 minutes. The sensing data were collected at a sampling rate of 50 Hz (i.e., one data point every 20 ms); thus, each print generated around 33,000 data points. These data were used to train TinyFR to extract the normal operational profile. Second, we collected data from 9 prints for each type of FWC attack. These datasets also include non-anomalous segments, as the attacks were initiated and terminated within specific layer ranges during the printing process. The anomalous segments are discussed in the next section. Finally, we extracted printer log data using the Prusa firmware [1], which includes built-in MK4 sensor readings sampled at 10 Hz. These log data support feature extraction and help to identify the current printing layer. Table 2 provides a detailed description of the dataset.

6.1.2 Data Preparation. Because the dataset was gathered over several days, the intervals between print jobs were inconsistent, making print start and end times difficult to identify from the sensing streams alone. We therefore used the MK4's internal log as ground truth: *syslog.state* indicates the printer's status, and *syslog.axis_z* records the vertical nozzle position in millimetres. Relying on *syslog.state* alone also captures non-printing movements such as nozzle cleaning, bed calibration, and the return to the home position (Figure 9(a)), whose duration varies with prior temperatures and filament type. To isolate the actual printing interval we extracted three values from *syslog.axis_z* (Figure 9(b)): the ***First Layer Height (FLH)*** of 0.2 mm, the ***Subsequent Layer Height (SLH)*** of 0.1 mm, and the ***Total Number of Layers***



(a) Attack duration of LT attack layers (b) Attack duration of VSP attack

Fig. 10. Data labelling process illustrated for two example attacks: (a) LT and (b) VSP. Upper plots show layer progression with attack boundaries (layers 90–200), while lower plots display sensor readings from all modalities across edge boards, labelled as anomalous (1) or non-anomalous (0) within the attack duration.

(TNL) of 234 (23.4 mm total height) for the bolt model. The characteristic descent to the FLH just before printing and the rise to the home position afterwards delimit the true printing duration, as shown in Figure 9(c).

6.1.3 Data Labelling. To label the dataset, we marked each sensor sample as anomalous if it occurred during the attack interval (i.e., within the attacked layers). For all anomalous prints described in Section 4.1, attacks start at layer 90 (corresponding to 9.0 mm on the Z-axis) and end at layer 200 (20.0 mm on the Z-axis). In our threat model, we assume that the attacker delays launching the attack until the middle of the print to reduce the likelihood of human detection, as supervision is more likely at the beginning and end of a print. Using these layer boundaries, we derived the corresponding start and end timestamps and used them to label sensor readings from all modalities. To determine the attack start and end times, we used the Z-axis (nozzle height) feature extracted from the printer log files in the previous step. This feature enabled consistent labelling across all edge boards. The analysis in Section 6.1.2 ensured that the data included only the actual printing duration, excluding irrelevant movements before and after the print. Figure 10(a) and Figure 10(b) illustrate the labelling process for two example attacks. Even within anomalous prints, there are segments of non-anomalous data. These clean segments enrich the test set and reduce the number of prints required during data collection.

6.1.4 Selected Features for Correlation Analysis. We extract features from the accelerometer signals over a sliding window of 486 samples with a 25-sample overlap, chosen via grid search to optimise detection performance. The **Instantaneous Resultant Acceleration (IRA)** is calculated as the Euclidean norm of the triaxial accelerations at each time step. For each axis (X, Y, Z), we calculate the **Root Mean Square (RMS)** and **kurtosis** within each window: RMS reflects the general intensity of motion and is commonly used in machine monitoring and fault detection [24], while kurtosis is sensitive to transient spikes linked to mechanical faults [2, 14]. From the IRA signal, we also extract the **Standard Deviation of Resultant Acceleration Peaks (STDRAP)** and the **Median of Resultant Acceleration Peaks (MEDRAP)**, which capture the variability and central tendency of vibration spikes, a known indicator of abnormal behaviour [15]. To examine inter-board signal similarity, we apply Pearson correlation coefficient (PCC) analysis to these features; as shown in Figure 11, the strong correlations across edge boards support the assumption

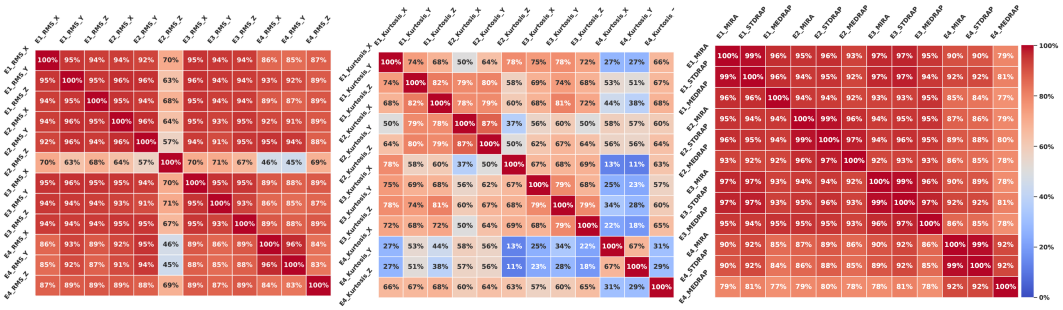


Fig. 11. Correlation matrices from triaxial acceleration for each edge board (E1–E4) computed using Root Mean Square (RMS), kurtosis, Instantaneous Resultant Acceleration (IRA), Standard Deviation of Resultant Acceleration Peaks (STDRAP), and Median of Resultant Acceleration Peaks (MEDRAP), showing relationships between sensor features.

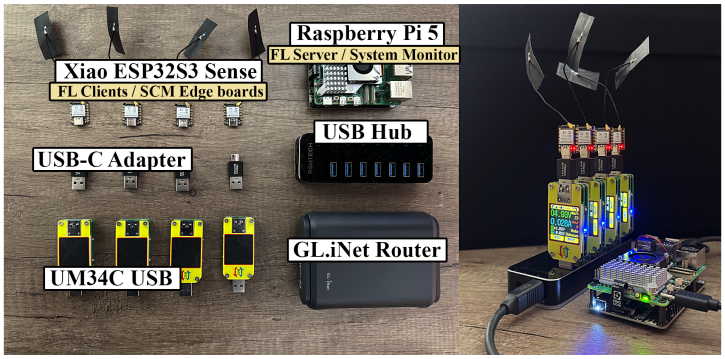


Fig. 12. TinyLearnED Runtime Experiment Setup

that all boards detect the same physical event during printing. We use this compact, interpretable feature set to train a separate TinyFR model for each board.

6.2 TinyFR Model Training and Deployment

We train the TinyFR model offline following a standard ML pipeline, and Figure 12 shows the runtime setup used to deploy and evaluate TinyLearnED on the edge boards. Owing to the distributed nature of TinyLearnED, each edge board is trained independently on its own dataset; the procedure is otherwise identical across boards. Datasets are split by print task using the 30 non-anomalous prints and 9 prints per FWC attack type outlined in Section 4.1. From the non-anomalous prints, 20 are used for training and 8 for validation while the remaining two are combined with the attack prints to form the test set, preventing data leakage. Training is performed on a local workstation equipped with an *NVIDIA RTX A6000 (48 GB GPU)*, *Intel Xeon W-2245 @3.90 GHz CPU*, and *128 GB RAM*. Each training run applies the sliding-window and preprocessing steps described in Section 5.4. We evaluate multiple lightweight ML models that are compatible with edge devices and support model compression, optimising each candidate's hyperparameters by grid search. Table 3 compares the candidates across edge boards: AE-LSTM achieves the strongest overall detection performance and is selected as the TinyFR backbone, then compressed and deployed to the edge boards. The edge board is a *XIAO ESP32S3 Sense*, running at 240 MHz with 512 KB DRAM, 8 MB PSRAM, and 8 MB Flash, and supporting both WiFi and ESPNOW. To handle memory constraints, we store the TinyFR model in external PSRAM, following a

Table 3. F1 score (%) of candidate anomaly-detection models per attack and edge board. Bold marks the best learned model per column; the Null Model is a majority-class reference. AE-LSTM achieves the strongest overall performance and is selected as the TinyFR backbone. Full accuracy, precision, recall, and false-positive rate (FPR) for all candidates are provided in the supplementary material.

	LT Attack				**VS Attack**				**VSP Attack**			
Model	E1	E2	E3	E4	E1	E2	E3	E4	E1	E2	E3	E4
Null Model	88.2	88.2	88.2	88.2	81.2	81.2	81.2	81.2	79.8	79.8	79.8	79.8
Moving Median	93.6	78.5	77.2	63.4	64.2	73.9	80.8	65.6	50.4	50.4	57.1	50.4
Isolation Forest	90.9	74.9	81.8	70.2	70.2	74.8	75.4	**74.1**	64.7	50.3	57.1	50.4
LSTM	96.2	88.8	88.9	75.7	**84.5**	88.4	95.9	66.1	**74.2**	51.3	76.3	**51.4**
1D-CNN	89.4	73.1	71.1	65.4	64.3	65.4	55.2	48.6	64.1	50.4	68.8	50.4
One-Class SVM	93.8	78.5	77.0	64.5	74.5	78.6	83.0	64.1	58.9	50.4	65.5	50.4
AE-Dense	91.9	73.6	74.4	47.7	61.4	66.2	76.3	55.5	50.9	50.4	59.6	50.4
AE-LSTM	**96.9**	**89.7**	**93.4**	**77.3**	83.3	**88.9**	**96.0**	72.9	72.7	**51.5**	**84.0**	51.3

strategy similar to [36]. Memory is statically allocated based on tensor sizes, buffers, and interpreter variables, avoiding the fragmentation and instability associated with dynamic allocation [17].

7 Evaluation and Ablation Analysis

This section evaluates TinyLearnED as a complete adaptive TinyML system under the deployment constraints of the XIAO ESP32S3 boards.

7.1 Evaluation Protocol, Baselines, and Metrics

This subsection defines the streaming evaluation protocol, the deployment modes compared, and how metrics are aggregated across the four edge boards.

7.1.1 Streaming evaluation protocol. We evaluate each method by replaying the test windows in chronological order for each edge board and each firmware-compromise attack scenario: Layer Thickness (LT), Void Skipping (VS), and Void Spoofing (VSP). For each window, the board first executes the TinyFR forward path and records its local anomaly decision. Depending on the deployment mode, it then exchanges binary peer decisions and computes a peer-consensus pseudo-label. In every TinyLearnED mode with OFT enabled, the local final layer is updated if and only if the recorded local decision disagrees with this pseudo-label, following Equation (4). This disagreement-based trigger was used throughout the reported OFT and FL+OFT experiments. Federated aggregation follows the configured interval rather than the local OFT trigger. The protocol remains strictly online: future windows are not used, and the test stream is never converted into an offline batch-training set. Ground-truth labels derived from the attack intervals are withheld during replay and used only afterwards to compute the reported metrics. TinyLearnED does not use these labels for pseudo-labelling, OFT, or FL.

7.1.2 Compared methods and baselines. We compare six deployment modes. *Inference Only* (*Static TinyFR*) keeps TinyFR frozen and performs no OFT, pseudo-labelling, or FL. We implemented a TinyOL-style local online-learning baseline following the last-layer adaptation principle of Ren et al. [28], adapted to the same TinyFR architecture and XIAO ESP32S3 platform used by TinyLearnED; it is our own implementation rather than a direct port of the original code. The baseline represents the unsupervised autoencoder setting: after predicting each incoming window, it treats that window as its reconstruction target and updates the final dense layer to reduce reconstruction error, without ground-truth labels, peer decisions, or federated aggregation. It updates on every incoming window with batch size 1 and uses the same Adam optimiser, learning rate, and optimiser settings as all TinyLearnED adaptive modes. Each comparison uses the same initial per-board TinyFR model, preprocessing, chronological stream order, data access,

Table 4. Aggregate edge-level performance (macro averages) across the four edge boards under different 3D-printing attacks. Our TinyOL-style baseline [28] performs local unsupervised reconstruction updates without peer consensus or federated aggregation.

Attack	Learning	Acc.	Prec.	Rec.	F1	FPR	Mode
LT	Inference Only	0.947	0.846	0.953	0.893	0.056	Static
	TinyOL-style baseline [28]	0.952	0.859	0.943	0.897	0.047	Adaptive
	OFT	0.959	0.868	0.955	0.908	0.040	Adaptive
	FL (Every 50 inf.)	0.952	0.862	0.952	0.903	0.049	Adaptive
	FL+OFT (Every 50 inf.)	0.936	0.822	0.871	0.845	0.049	Adaptive
	FL+OFT (Every 150 inf.)	0.967	0.889	0.963	0.924	0.033	Adaptive
VS	Inference Only	0.897	0.800	0.917	0.853	0.113	Static
	TinyOL-style baseline [28]	0.920	0.858	0.895	0.875	0.069	Adaptive
	OFT	0.928	0.872	0.904	0.888	0.062	Adaptive
	FL (Every 50 inf.)	0.907	0.822	0.921	0.867	0.100	Adaptive
	FL+OFT (Every 50 inf.)	0.900	0.812	0.886	0.846	0.095	Adaptive
	FL+OFT (Every 150 inf.)	0.938	0.884	0.927	0.904	0.058	Adaptive
VSP	Inference Only	0.601	0.524	0.941	0.649	0.573	Static
	TinyOL-style baseline [28]	0.590	0.512	0.859	0.617	0.546	Adaptive
	OFT	0.600	0.521	0.949	0.650	0.578	Adaptive
	FL (Every 50 inf.)	0.605	0.532	0.948	0.656	0.571	Adaptive
	FL+OFT (Every 50 inf.)	0.694	0.498	0.868	0.631	0.380	Adaptive
	FL+OFT (Every 150 inf.)	0.602	0.540	0.897	0.641	0.551	Adaptive

hardware, and reset procedure. The shared online-learning parameters are fixed across the adaptive modes rather than tuned separately for the TinyOL-style baseline. They are selected without final test labels; the normal-only validation data described in Section 6.2 are used for the TinyFR preprocessing, model, and threshold choices, not to optimise anomaly F1 for this baseline. It is the only framework critiqued in Section 3 that can operate in our setting: LifeLearner requires a labelled replay buffer and TinyTL assumes post-deployment ground-truth labels, neither of which exists in our label-free streaming scenario. *OFT* enables peer-consensus pseudo-labels and local final-layer fine-tuning without federating the updated parameters. *FL only* periodically averages final-layer parameters without local OFT, isolating the stabilising effect of parameter sharing. The remaining modes, *FL+OFT every 50 inferences* and *FL+OFT every 150 inferences*, run the full TinyLearnED pipeline and contrast frequent aggregation with lower communication cost.

7.1.3 Metrics and aggregation. We treat anomalous windows as the positive class and report accuracy, precision, recall, F1 score, and false-positive rate (FPR). Aggregate results are unweighted macro averages across the four edge boards: each metric, including F1, is computed per board and then averaged rather than derived from pooled counts. Accuracy is reported for completeness but is not the primary comparison metric, because the stream is imbalanced: anomalous windows occur only during specific attacked layers, while normal windows dominate the print stream. We therefore emphasise F1 and FPR. High recall alone can mask excessive false alarms, and false positives are operationally costly in continuous SCM deployments because they can trigger unnecessary inspection, process interruption, or rejection of printed objects. Per-edge metric tables are provided in the supplementary material.

7.2 Aggregate Comparison with Static TinyFR and the TinyOL-style Baseline

Table 4 compares TinyLearnED with a static TinyFR deployment and the TinyOL-style baseline defined in Section 7.1.2. The comparison controls the initial per-board model, frozen TinyFR forward path, trainable output-layer capacity, preprocessing, stream order, and hardware. It is not a single-factor isolation of collaboration, because the TinyOL-style baseline directly minimises reconstruction error on every incoming window, whereas TinyLearnED uses disagreement-gated, BCE-derived surrogate updates with peer-consensus pseudo-labels, as specified in Algorithm 1. For the LT attack, Static TinyFR achieves an aggregate F1 score of 0.893 with an FPR of 0.056. The TinyOL-style baseline slightly improves F1 to 0.897 and reduces FPR to 0.047, showing that local online adaptation can provide some benefit when the physical anomaly is visible in the vibration stream. However, TinyLearnED provides larger gains when peer-consensus

Table 5. Per-edge F1 score and FPR (anomaly as the positive class) for each deployment mode. R denotes the federated aggregation interval in inference windows. Bold marks the best F1 per row. Full per-edge accuracy, precision, and recall, together with per-mode deltas against the static baseline, are provided in the supplementary material.

Attack	Edge	Static	OFT	FL (R=50)	FL+OFT (R=50)	FL+OFT (R=150)
		F1 / FPR	F1 / FPR	F1 / FPR	F1 / FPR	F1 / FPR
LT	E1	0.969 / 0.013	0.969 / 0.013	**0.976** / 0.010	0.909 / 0.029	**0.976** / 0.010
	E2	0.897 / 0.047	0.896 / 0.045	0.907 / 0.043	0.915 / 0.040	**0.923** / 0.035
	E3	0.934 / 0.023	0.933 / 0.022	0.937 / 0.021	0.816 / 0.048	**0.947** / 0.020
	E4	0.773 / 0.140	0.834 / 0.080	0.791 / 0.120	0.741 / 0.079	**0.848** / 0.065
VS	E1	0.833 / 0.114	0.881 / 0.059	0.850 / 0.102	0.862 / 0.094	**0.894** / 0.051
	E2	0.889 / 0.089	0.894 / 0.064	0.909 / 0.060	0.802 / 0.091	**0.917** / 0.055
	E3	0.960 / 0.029	0.960 / 0.024	0.967 / 0.024	**0.972** / 0.015	0.967 / 0.020
	E4	0.729 / 0.219	0.815 / 0.101	0.742 / 0.214	0.746 / 0.178	**0.839** / 0.104
VSP	E1	0.727 / 0.270	0.730 / 0.280	**0.738** / 0.265	0.668 / 0.327	0.729 / 0.234
	E2	0.515 / 0.947	0.515 / 0.946	0.519 / 0.948	**0.574** / 0.528	0.517 / 0.937
	E3	0.840 / 0.130	0.840 / 0.140	**0.852** / 0.122	0.661 / 0.302	0.801 / 0.093
	E4	0.513 / 0.944	0.513 / 0.944	0.516 / 0.950	**0.621** / 0.363	0.516 / 0.939

pseudo-labels and federated smoothing are used appropriately. OFT improves F1 to 0.908, FL alone improves F1 to 0.903, and the full FL+OFT configuration with aggregation every 150 inferences achieves the best LT result, with an F1 score of 0.924 and the lowest FPR of 0.033. The FL+OFT configuration with aggregation every 50 inferences performs worse on LT, indicating that more frequent aggregation can propagate noisy early updates before local evidence has stabilised. The same pattern is clearer under the VS attack. Static TinyFR reaches an aggregate F1 score of 0.853, while the TinyOL-style baseline improves to 0.875. OFT further improves F1 to 0.888 by using peer-consensus pseudo-labels rather than relying only on local online adaptation. The best VS result is again obtained by FL+OFT with aggregation every 150 inferences, which reaches an F1 score of 0.904 and reduces FPR to 0.058. These results show that TinyLearnED improves over local-only online learning when the shared physical signal is sufficiently strong across the SCM boards. In this setting, the TinyOL-style baseline adapts each board independently, whereas TinyLearnED uses peer agreement to gate adaptation and federated averaging to smooth final-layer parameters across boards. The VSP attack remains substantially harder. Static TinyFR obtains an aggregate F1 score of 0.649, while the TinyOL-style baseline decreases to 0.617, mainly because direct adaptation to every incoming window reduces recall without a reliable external update signal. OFT and FL alone provide only marginal improvements, with F1 scores of 0.650 and 0.656, respectively. The full FL+OFT configuration with aggregation every 50 inferences substantially reduces FPR from 0.573 to 0.380, but its F1 score remains 0.631 because the recall–precision balance remains weak. This behaviour is consistent with the attack mechanism: VSP preserves nozzle motion while suppressing extrusion, so the vibration evidence is less distinctive than in LT or VS. We therefore do not claim that TinyLearnED fully resolves VSP detection. Instead, the results show that TinyLearnED reduces some failure modes, particularly false alarms under frequent aggregation, but VSP remains the most challenging attack scenario for accelerometer-only SCM.

7.3 Component Ablation of TinyLearnED

Table 5 decomposes the aggregate results by edge board and deployment mode. This ablation identifies which components drive the observed performance changes. While Table 4 provides the overall comparison with Static TinyFR and the TinyOL-style baseline, Table 5 shows how OFT, FL, and aggregation frequency behave across different sensor placements and attack signatures.

7.3.1 Effect of online fine-tuning. OFT improves performance when the static model is imperfect but the peer-consensus pseudo-label provides a useful correction signal. Under the LT attack, OFT leaves the already strong boards nearly unchanged: edge1 remains at an F1 score of 0.969, while edge2 and edge3 change only from 0.897 to 0.896 and from 0.934 to 0.933, respectively. The largest LT improvement occurs on edge4, where F1 increases from 0.773 to 0.834 and FPR decreases from 0.140 to 0.080. This shows that OFT is most useful when the local static model is weaker but the collaborative signal remains informative. Under the VS attack, the same pattern appears more consistently: edge1 improves from 0.833 to 0.881, edge2 from 0.889 to 0.894, edge4 from 0.729 to 0.815, while edge3 remains stable at 0.960 because its static TinyFR model is already strong. In contrast, OFT has little effect under VSP: edge1 improves only from 0.727 to 0.730, edge2 remains at 0.515, edge3 remains at 0.840, and edge4 remains at 0.513. These results indicate that OFT is not a generic performance booster; it depends on both pseudo-label quality and the detectability of the physical attack signature.

7.3.2 Effect of federated smoothing. FL alone provides modest but consistent stabilisation by sharing final-layer parameters across boards without activating peer-consensus-driven local OFT. Under LT, FL every 50 inferences improves F1 on all four boards: edge1 increases from 0.969 to 0.976, edge2 from 0.897 to 0.907, edge3 from 0.934 to 0.937, and edge4 from 0.773 to 0.791. Under VS, FL also improves all boards, increasing F1 from 0.833 to 0.850 on edge1, from 0.889 to 0.909 on edge2, from 0.960 to 0.967 on edge3, and from 0.729 to 0.742 on edge4. Under VSP, the gains are smaller but remain positive, with F1 increasing from 0.727 to 0.738 on edge1, from 0.515 to 0.519 on edge2, from 0.840 to 0.852 on edge3, and from 0.513 to 0.516 on edge4. These results show that FL is not merely a communication add-on. It acts as a lightweight alignment mechanism across boards that observe related, but not identical, vibration profiles. However, because FL alone does not create new local evidence or correct weak pseudo-labels, its gains remain smaller than the best FL+OFT configuration for LT and VS.

7.3.3 Effect of aggregation frequency. The comparison between FL+OFT every 50 inferences and FL+OFT every 150 inferences shows that more frequent aggregation is not automatically better. For LT, aggregation every 150 inferences outperforms aggregation every 50 inferences on all boards except where both settings are close: edge1 improves from 0.909 to 0.976, edge2 from 0.915 to 0.923, edge3 from 0.816 to 0.947, and edge4 from 0.741 to 0.848. For VS, the 150-inference interval is also more stable overall: edge1 improves from 0.862 to 0.894, edge2 from 0.802 to 0.917, and edge4 from 0.746 to 0.839, while edge3 remains high under both settings. These results suggest that aggregation every 50 windows can share local updates quickly, but it can also propagate noisy pseudo-label-driven updates before enough evidence accumulates at each board. Aggregation every 150 windows reduces this effect and often improves stability for LT and VS. The VSP case shows a different trade-off. Aggregation every 50 inferences improves some weak boards relative to the 150-inference setting, but it does not produce the best aggregate F1 because precision and recall remain limited. This confirms that aggregation frequency is a deployment parameter rather than a universal optimum. A smaller aggregation interval favours faster sharing and stronger false-alarm suppression in some cases, but increases communication cost and can spread noisy updates. A larger interval reduces communication overhead and improves stability when pseudo-label quality is uneven. Section 8.3 discusses why VSP is especially sensitive to physical observability.

Table 6. TinyLearnED performance under the SCMC (defect-hiding) attack, FL+OFT, 50 inferences/update.

Status	C0 (No SCMC)					C1					C2					C3				
	Acc %	Pre %	Rec %	F1 %	FPR %	Acc %	Pre %	Rec %	F1 %	FPR %	Acc %	Pre %	Rec %	F1 %	FPR %	Acc %	Pre %	Rec %	F1 %	FPR %
LT Attack	94	82	87	85	5	$91_{\downarrow 3}$	$79_{\downarrow 3}$	$80_{\downarrow 7}$	$79_{\downarrow 6}$	$4_{\downarrow 1}$	$81_{\downarrow 13}$	$50_{\downarrow 32}$	$12_{\downarrow 75}$	$19_{\downarrow 66}$	$3_{\downarrow 2}$	$79_{\downarrow 15}$	$30_{\downarrow 52}$	$3_{\downarrow 84}$	$5_{\downarrow 80}$	$1_{\downarrow 4}$
VS Attack	90	81	89	85	9	$86_{\downarrow 4}$	$77_{\downarrow 4}$	$80_{\downarrow 9}$	$78_{\downarrow 7}$	$7_{\downarrow 2}$	$70_{\downarrow 20}$	$48_{\downarrow 33}$	$14_{\downarrow 75}$	$22_{\downarrow 63}$	$4_{\downarrow 5}$	$68_{\downarrow 22}$	$28_{\downarrow 53}$	$4_{\downarrow 85}$	$7_{\downarrow 78}$	$2_{\downarrow 7}$
VSP Attack	69	50	87	63	38	$66_{\downarrow 3}$	$40_{\downarrow 10}$	$55_{\downarrow 32}$	$46_{\downarrow 17}$	$30_{\downarrow 8}$	$65_{\downarrow 4}$	$32_{\downarrow 18}$	$20_{\downarrow 67}$	$25_{\downarrow 38}$	$15_{\downarrow 23}$	$65_{\downarrow 4}$	$22_{\downarrow 28}$	$7_{\downarrow 80}$	$11_{\downarrow 52}$	$6_{\downarrow 32}$

7.4 Robustness under SCMC Attacks

This subsection evaluates TinyLearnED under Side-Channel Monitoring Node Compromise (SCMC), where the adversary attacks the monitoring infrastructure to hide defects produced by the compromised printing process. Table 6 reports the robustness results for the full TinyLearnED configuration under increasing numbers of compromised edge boards.

7.4.1 SCMC evaluation setting. The SCMC experiment evaluates a defect-hiding adversary. In this setting, the 3D printer is already subject to one of the firmware-compromise attacks, namely LT, VS, or VSP, and the adversary additionally compromises a subset of the SCM edge boards to suppress evidence of the defect. We evaluate four compromise levels: C0 denotes no compromised SCM node, C1 denotes one compromised edge board, C2 denotes two compromised edge boards, and C3 denotes three compromised edge boards. A compromised board forces its transmitted decision to the benign class, i.e., it reports normal even when its local TinyFR path would identify an anomalous window. Because the evaluated configuration uses FL+OFT, the compromise can also affect the trainable final-layer parameters uploaded by that board during federated averaging. The adversary does not alter the raw sensor windows or model state of honest boards; instead, it corrupts the collaborative decision channel and the local update contribution of compromised boards. This setting captures the intended SCMC threat model: the attacker attempts to make genuine printing defects appear benign rather than injecting false alarms.

7.4.2 Robustness results under increasing compromise. The SCMC table shows a bounded robustness pattern. With no compromised SCM node (C0), TinyLearnED achieves strong detection for LT and VS, with F1 scores of 85% in both cases. VSP is weaker at C0, with an F1 score of 63%, which is consistent with the earlier aggregate results showing that VSP is the hardest attack because nozzle motion is preserved while extrusion is suppressed. With one compromised node (C1), the system remains partially effective for LT and VS: LT F1 decreases from 85% to 79%, and VS F1 decreases from 85% to 78%. The corresponding recalls decrease from 87% to 80% for LT and from 89% to 80% for VS, indicating that the remaining honest boards still provide enough evidence to sustain useful detection for stronger physical signatures. In contrast, VSP degrades more sharply at C1, with F1 falling from 63% to 46% and recall from 87% to 55%. This confirms that VSP has a smaller robustness margin; Section 8.3 discusses the physical-observability reasons for this behaviour. The degradation becomes non-linear once the adversary controls half of the monitoring network. At C2, LT recall collapses from 87% to 12%, and F1 drops from 85% to 19%. VS follows the same pattern, with recall falling from 89% to 14% and F1 from 85% to 22%. VSP also degrades severely, reaching 20% recall and 25% F1. At C3, the detector is close to failure: recall falls to 3% for LT, 4% for VS, and 7% for VSP, with F1 scores of 5%, 7%, and 11%, respectively. These results are consistent with the consensus-rule analysis: the pseudo-label remains reliable only while honest nodes retain a strict voting advantage. Once two of four boards are compromised, the honest boards no longer provide a reliable majority signal, and the collaborative labelling mechanism can no longer distinguish genuine defects from adversarially suppressed reports. The FPR trend should therefore be interpreted in the context of a defect-hiding attack. As more boards are compromised, FPR falls (LT: 5% at C0 to 1% at C3; VSP: 38% to 6%). This does not indicate that the system is becoming safer; rather, the detector is becoming silent. SCMC suppresses anomaly decisions, creating false negatives

Table 7. Analytical resource model of TinyLearnED. All quantities derive from the TinyFR architecture, OFT trainable layer, and communication payload format.

Quantity	Value	Derivation / meaning
Model size		
TinyFR parameters	4,811	Full AE-LSTM reconstruction model.
Trainable parameters	1,323 (27.5%)	Final dense layer only: $27 \times 48 + 27$.
Frozen parameters	3,488 (72.5%)	Encoder–decoder parameters not updated on-device.
Arithmetic per inference		
MACs	11,280	LSTM and dense matrix-vector operations, $T = 3$.
FLOPs	22,560	2× MACs.
Storage and optimiser state		
TinyFR model storage	19.2 / 9.6 KB	FP32 / FP16 storage for 4,811 parameters.
Trainable layer storage	5.3 KB	$1,323 \times 4$ bytes in FP32.
Adam state (m, v)	10.6 KB	Two FP32 moment buffers for 1,323 parameters.
TFLM tensor arena	64 KB	Static interpreter arena allocated in PSRAM.
Communication payload		
Peer vote application payload	16 bytes/message	Node id, window id, timestamp, binary decision, status, and checksum; transport headers excluded.
Peer consensus application payload	96 bytes/window	$2(N-1)B_{\text{msg}}$, with $N = 4$ and $B_{\text{msg}} = 16$ bytes.
FL round parameter payload	10,584 bytes/client	One 5,292-byte upload and one 5,292-byte download; protocol overhead excluded.
Amortised FL payload, $R = 50$	212 bytes/window	$2B_{\text{param}}/50$.
Amortised FL payload, $R = 150$	71 bytes/window	$2B_{\text{param}}/150$.
Total application payload	308 / 167 bytes/window	Fixed peer exchange plus amortised FL for $R = 50$ / $R = 150$; retransmissions excluded.

Table 8. Measured runtime cost on XIAO ESP32S3 at 240 MHz. Runtime values are reported over the processed TinyFR windows generated by the feature-transformation pipeline. Peak memory reports application-level memory after Wi-Fi/ESPNOW initialisation. The TinyOL-style baseline row measures the same reimplementation used for the detection comparison in Table 4.

Execution path	Mean (ms)	Std. (ms)	Current (mA)	Energy (mJ)	Peak mem. (KB)
TinyFR inference only	6.5	0.4	118	2.53	94
Static TinyFR path	9.3	0.7	121	3.71	94
TinyOL-style baseline [28]	13.1	0.9	128	5.53	111
Consensus, no OFT	11.9	0.9	137	5.38	96
Consensus + OFT (ours)	15.7	1.2	145	7.51	111
Consensus + OFT (FreeRTOS no-affinity)	21.4	4.8	146	10.31	111

rather than false positives, and the paired decrease in FPR and recall indicates successful defect hiding. Accuracy has the same limitation under class imbalance: for VSP, it remains nearly flat (69% to 65%) while recall collapses from 87% to 7%. Therefore, recall and F1 are the primary robustness indicators under SCMC. FL+OFT also has two roles under compromise. Under benign or limited-compromise conditions, FL smooths noisy final-layer updates and allows weaker boards to benefit from stronger peers. Under SCMC, however, the same averaging path propagates suppression-biased parameters from compromised boards, making the C1–C2 transition sharper than a vote-only interpretation would suggest. TinyLearnED therefore provides robustness under an honest-majority assumption rather than Byzantine fault tolerance. It tolerates limited SCMC for attacks with strong shared physical signatures, but degrades rapidly once half or more of the SCM network is compromised.

7.5 Runtime, Memory, Energy, and Communication Cost

This subsection quantifies the resource overhead of TinyLearnED on the target XIAO ESP32S3 platform. We analyse the cost from two complementary views. First, Table 7 reports the model footprint, trainable state, arithmetic cost, and

communication payloads derived from the TinyFR architecture and protocol format. Second, Table 8 reports measured latency, current, active energy, and peak memory for the main execution paths. Together, these results show that TinyLearnED adds measurable overhead compared with static inference and the TinyOL-style baseline, but keeps both computation and communication bounded by updating only the final layer and amortising FL communication over multiple inference windows.

7.5.1 Analytical resource footprint. The analytical resource table shows that TinyLearnED keeps on-device learning small by freezing most of TinyFR. The full TinyFR model contains 4,811 parameters, but only the final dense layer is trainable. This layer contains 1,323 parameters, computed as $27 \times 48 + 27$, representing 27.5% of the model. The remaining 3,488 parameters, or 72.5%, belong to the frozen encoder–decoder path and are never updated on the MCU. This design limits both memory and computation during adaptation: the trainable layer requires 5.3 KB in FP32, and the Adam optimiser state requires 10.6 KB for the two moment buffers. The TFLite Micro tensor arena is statically allocated at 64 KB, while TinyFR model storage is 19.2 KB in FP32 or 9.6 KB in FP16. The inference path requires 11,280 MACs, or approximately 22,560 FLOPs, which is consistent with the lightweight AE-LSTM and final dense reconstruction layer. These values explain why TinyLearnED can support online updates on the XIAO ESP32S3 without full-model backpropagation or cloud retraining. Communication cost is bounded by separating the 16-byte peer-decision application payload from the larger FL parameter exchange. With four boards, each board sends to and receives from three peers, giving a fixed peer-exchange application payload of 96 bytes per window. Transport headers and retransmissions are not included in this accounting. Each FL direction carries 5,292 parameter bytes, so one upload and one download total 10,584 bytes per client and aggregation round before protocol overhead. Amortisation contributes approximately 212 bytes per window for $R = 50$ and 71 bytes per window for $R = 150$. Adding the fixed peer component gives total application payloads of approximately 308 and 167 bytes per window, respectively. Thus, increasing R reduces only the amortised FL term, while the 96-byte peer contribution remains constant.

7.5.2 Measured MCU runtime and scheduler overhead. The measured runtime table reports the cost of each execution path on the XIAO ESP32S3 at 240 MHz. Current was measured using a UM34C meter while the firmware repeatedly executed a sustained loop for each path; the table reports the mean current per loop. Active energy is computed as $E = V \cdot I \cdot t$ using the measured mean current, the 3.3 V supply voltage, and the mean execution time. TinyFR inference alone takes 6.5 ms per window, and the full static TinyFR path takes 9.3 ms including the surrounding execution logic. The same TinyOL-style baseline implementation evaluated in Table 4 takes 13.1 ms with a peak memory footprint of 111 KB, while the full TinyLearnED adaptive path with consensus and OFT takes 15.7 ms (standard deviation 1.2 ms) at the same 111 KB peak. The collaborative decision channel and its surrounding coordination therefore add 2.6 ms per adaptive window over the TinyOL-style baseline. Average current rises from 128 mA to 145 mA, corresponding to active powers of approximately 0.42 W and 0.48 W at 3.3 V, and active energy increases from 5.53 mJ to 7.51 mJ. This overhead is measurable but remains within the timing and memory budget of continuous window-level monitoring on the target MCU. The scheduler comparison demonstrates the practical value of the custom FreeRTOS task assignment described earlier: compared with the pinned configuration, no-affinity scheduling increases the mean runtime of the same consensus+OFT workload from 15.7 ms to 21.4 ms, runtime variability from 1.2 ms to 4.8 ms, and active energy from 7.51 mJ to 10.31 mJ, while peak memory remains unchanged at 111 KB. Task migration and less predictable core placement therefore introduce scheduling overhead, even though the learning algorithm itself is unchanged. Pinning inference, collaboration, learning, and communication tasks to appropriate cores improves both latency and determinism.

This matters for TinyLearnED, which must coordinate sensing, feature extraction, inference, peer exchange, OFT, and FL participation on a single MCU without delaying the streaming monitoring loop.

7.5.3 Aggregation Frequency Trade-off. The aggregation interval R controls a direct trade-off between communication cost, adaptation speed, and stability. The analytical resource table shows that increasing R from 50 to 150 reduces total amortised communication from 308 to 167 bytes/window, a reduction of approximately 46%. The aggregate performance table shows that this lower-cost setting is also more stable for the stronger LT and VS attack signatures: FL+OFT every 150 inferences achieves the best aggregate F1 for both attacks while also reducing FPR relative to the 50-inference setting. Frequent aggregation is therefore not inherently beneficial: when pseudo-labels are noisy, rapid sharing can propagate unstable local updates before each board has accumulated sufficient evidence. For VSP the trade-off differs: aggregation every 50 inferences reduces FPR more strongly, but F1 remains limited by the attack's inherently weak vibration evidence. R should therefore be treated as a deployment parameter: smaller values favour faster sharing and stronger correction of some false-alarm modes, while larger values reduce communication and can improve stability when the physical anomaly signature is clear. In practice, R should be chosen jointly with the expected attack type, communication budget, and tolerance for false alarms.

8 Discussion

The evaluation shows that TinyLearnED supports weakly supervised online adaptation on constrained SCM edge boards, but it is not a plug-and-play anomaly detector that transfers unchanged across CPSs, object geometries, sensor placements, or attack types. Its effectiveness depends on three deployment properties: the frozen TinyFR representation must capture the relevant normal-operation dynamics, the participating SCM nodes must observe correlated physical events, and the pseudo-labels generated by peer consensus must remain sufficiently reliable for OFT.

8.1 Scope of Generalisability Beyond the Bolt 3D Testbed

Our evaluation uses a real CPS testbed, real XIAO ESP32S3 boards, and physically collected vibration data, but its empirical scope is bounded: the Bolt 3D dataset covers one Prusa MK4 printer, one printed object geometry, and one sensing modality, triaxial acceleration. Therefore, the results provide realistic evidence for the evaluated additive-manufacturing setting. They should not be interpreted as showing that the current TinyFR configuration will transfer unchanged to different printers or to new geometries, materials, and IoT domains. This single-modality design is a key limitation: accelerometer-only sensing restricts detection to attacks with observable vibration signatures and places substantial reliance on the feature-extraction pipeline and selected AD model. As shown in Table 3, RNN-based models generally outperform non-recurrent alternatives in our setting, as expected for time-dependent data and consistent with prior SCM work using AE–LSTM models on 3D-printer vibration data [33]. Vibration alone, however, is not sufficient for all defect classes; more robust SCM deployments may require heterogeneous nodes with additional sensing modalities (Section 8.3). The main transfer risk is the frozen TinyFR encoder. Updating only the final reconstruction layer makes OFT feasible on an MCU, but it requires the latent representation to already encode the relevant temporal structure of the monitored process. A different object geometry, toolpath, infill pattern, printer frame, mounting location, or material-flow behaviour may produce vibration profiles outside the Bolt 3D training distribution, in which case last-layer adaptation alone may be insufficient because the missing information is not present in the frozen latent features. TinyLearnED should then be re-instantiated by collecting normal-operation data from the new process, retraining or recalibrating TinyFR offline, and redeploying OFT and FL for post-deployment adaptation. The generality of TinyLearnED therefore lies in the system architecture rather than in a universal trained model: the

collaborative pipeline applies to other CPS monitoring tasks when multiple edge nodes observe related manifestations of the same physical process, the local feature extractor is trained on representative normal behaviour, and the anomaly threshold is calibrated for the target deployment. Future evaluations should include multiple print geometries, printer models, mounting locations, and heterogeneous sensing modalities. Correlation analysis is consequently a deployment prerequisite: we rely on Pearson correlation coefficient (PCC) analysis to select boards that observe related printing events, but distance-based or information-theoretic techniques could serve equally well [12], since the requirement is not PCC specifically but evidence that participating nodes observe sufficiently related manifestations of the same CPS event. A new deployment should group candidate boards by shared event observability, verify temporal alignment, and select the consensus threshold from the number of informative boards rather than the nominal device count. Robust deployments should additionally use authenticated peer messages, careful sensor placement, and heterogeneous sensing where possible. This aligns with Gao et al.'s 3D-printer monitoring work [12], which notes that magnetic-field calibration for DoF sensing should be adjusted to the target application. TinyLearnED follows the same principle, in which thresholds, sensing setup, and ML model selection are deployment-specific rather than universal.

8.2 Threshold Calibration and Deployment Transfer

The anomaly threshold τ is calibrated offline from the TinyFR training data and converts reconstruction error into a binary anomaly decision during deployment. It also enters the differentiable OFT loss, where reconstruction error is mapped to an anomaly likelihood through a sigmoid transformation, so τ affects both detection and adaptation. If τ is too low, normal operating variation can produce false anomaly decisions and noisy pseudo-labels; if it is too high, subtle defects may fail to trigger detection or OFT. We therefore treat τ not as a universal constant but as a deployment parameter tied to the trained TinyFR model, sensor mounting, mechanical environment, preprocessing pipeline, and expected normal operating envelope. In practice, τ should be set during pre-deployment using a short normal-operation calibration stream from the target CPS. This requires representative benign operation but no labelled attack data. A conservative implementation can select τ from the upper tail of the normal reconstruction-error distribution according to the deployment's tolerance for false alarms. Safety-critical deployments may prefer a lower threshold to improve sensitivity, and maintenance-oriented deployments a higher one to reduce unnecessary interruptions. Threshold transfer should also be monitored after deployment, since environmental changes, sensor remounting, printer maintenance, or long-term mechanical wear can shift the reconstruction-error distribution. The present evaluation does not quantify tolerance to sustained drift; a large threshold mismatch could corrupt the pseudo-label stream before adaptation becomes useful. Practical deployments should therefore include threshold health checks, such as monitoring reconstruction-error statistics during consensus-normal periods, bounding the threshold update rate, and reverting to a validated threshold when peer agreement becomes unstable.

8.3 Limits under Stealthy Physical Perturbations

Detecting attacks is challenging when adversarial actions result in only minor deviations from normal cyber-physical system (CPS) operation [20]. LT and VS generate more distinct vibration signatures because they alter layer formation or omit segments of the infill process. In contrast, VSP is designed to be stealthier by suppressing extrusion while maintaining nozzle trajectories, resulting in a weaker vibration footprint. This limitation is consistently observed in the evaluation: edge 4 underperforms compared to other boards in several scenarios due to its greater distance from the primary vibration source. Edge 3, positioned directly adjacent to the nozzle cooling fan, detects more pronounced vibration changes when extrusion is intermittently halted between layers 90 and 200. In comparison, edge 1, although also located near the nozzle head, provides weaker VSP evidence due to its top-mounted configuration. These placement

effects indicate that TinyLearnED's effectiveness depends not only on model architecture but also on the physical observability of the process. The system demonstrates improved adaptation and mitigates certain failure modes for LT and VS, where the shared physical signal is stronger. However, for VSP, performance remains limited by reliance on accelerometer-only observability, as online adaptation cannot generate discriminative evidence that is absent or weak in the sensed modality. Detecting such stealthy perturbations may require integrating additional extrusion-aware signals, such as filament-feed motion, motor current, nozzle pressure, acoustic emission, thermal behaviour, or visual inspection. TinyLearnED can incorporate these as supplementary SCM nodes or local features, although the current evaluation does not assess multimodal performance.

8.4 Catastrophic Forgetting and Long-Term OFT

Online machine learning is susceptible to catastrophic forgetting, in which sequential updates gradually overwrite previously acquired knowledge; LSTM networks are similarly affected [10]. In TinyLearnED, the LSTM-based TinyFR encoder remains static, and OFT modifies only the final reconstruction layer. This approach restricts adaptation and reduces MCU overhead, but does not fully prevent forgetting in the trainable layer. The current evaluation demonstrates short-term online adaptation using chronological replay of controlled printing streams. However, it does not establish multi-month stability under conditions such as changing materials, mechanical wear, sensor remounting, maintenance events, or prolonged adversarial manipulation. FL reduces local update noise by averaging final-layer parameters across boards, but this is not a comprehensive strategy for mitigating forgetting. Under SCMC, the same averaging process can propagate biased updates from compromised boards to legitimate ones, accelerating degradation if the legitimate-majority assumption fails. Practical MCU-compatible safeguards include limiting the OFT learning rate, bounding updates per window, retaining a validated final-layer snapshot for rollback, monitoring reconstruction-error drift, and maintaining a small trusted replay buffer or prototype summary when memory permits. These measures are proposed as future extensions rather than assumptions of the current evaluation. The present work introduces a lightweight adaptive pipeline validated on a real MCU-based CPS testbed, with long-term continual learning stability identified as a direction for future deployment studies.

9 Conclusion and Future Work

Resource-constrained IoT devices function within dynamic environments in which concept drift progressively reduces the accuracy of static pre-trained models, whereas cloud-based retraining and raw data transmission are impractical due to cost, connectivity, and privacy constraints. The TinyLearnED framework addresses these challenges by integrating peer-consensus pseudo-labelling, last-layer online fine-tuning, and lightweight federated aggregation on MCU-based edge boards, exchanging only binary anomaly decisions and final-layer parameters. In the evaluated single-printer, single-geometry, accelerometer-based testbed, four XIAO ESP32S3 monitoring nodes process chronological replays of physically collected Bolt 3D streams. With aggregation every $R = 150$ inferences, the complete pipeline increases the aggregate F1 score compared to a static TinyFR deployment from 0.893 to 0.924 for LT and from 0.853 to 0.904 for VS. It also outperforms the TinyOL-style baseline for these attacks; under the stealthier VSP attack, the TinyOL-style baseline performs worse than the static model. For VSP, a different configuration with aggregation every $R = 50$ inferences reduces aggregate FPR from 0.573 to 0.380, although the aggregate F1 score remains limited. TinyLearnED retains useful detection under the tested single-node compromise for attacks exhibiting strong shared physical signatures, whereas performance degrades when two or more of the four monitoring nodes are compromised, consistent with the K-of-N analysis. The complete adaptive pipeline remains within MCU resource constraints. Future research directions include

dynamic peer-relationship discovery to replace manual correlation analysis, federated knowledge distillation, evaluation across additional print geometries, printer models, and sensing modalities, and strengthening the collaboration channel through authenticated peer messaging. These findings are bounded to this testbed. Transfer to other printers, geometries, and sensing modalities, together with sustained long-term adaptation, requires further evaluation.

Acknowledgments

This work has been partially supported by the PETRAS National Centre of Excellence for IoT Systems Cybersecurity (Resilient Built Environments (ResBE) Project) EP/S035362/1, Digital Catapult Researcher-in-Residence program (EP/T517203/1), Innovate UK CyberASAP, GCHQ National Resilience Fellowship, and EPSRC National Edge Artificial Intelligence Hub (EP/Y028813/1). The authors would like to acknowledge the scholarship and support provided by King Abdulaziz University, Jeddah, Saudi Arabia.

References

[1] Prusa3D 2025. *Prusa3d/Prusa-Firmware-Buddy*. Prusa3D. https://github.com/prusa3d/Prusa-Firmware-Buddy

[2] Changbum R. Ahn, SangHyun Lee, and Feniosky Peña-Mora. 2015. Application of Low-Cost Accelerometers for Measuring the Operational Efficiency of a Construction Equipment Fleet. *Journal of Computing in Civil Engineering* 29, 2 (2015), 04014042. doi:10.1061/(ASCE)CP.1943-5487.0000337

[3] Abdulaziz Aljohani, Omer Rana, and Charith Perera. 2025. Self-adaptive Federated Learning in Internet of Things Systems: A Review. *ACM Comput. Surv.* 57, 10, Article 252 (May 2025), 36 pages. doi:10.1145/3725527

[4] Christian Bayens, Tuan Le, Luis Garcia, Raheem Beyah, Mehdi Javanmard, and Saman Zonouz. 2017. See No Evil, Hear No Evil, Feel No Evil, Print No Evil? Malicious Fill Patterns Detection in Additive Manufacturing. 1181–1198. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/bayens

[5] Sofia Belikovetsky, Mark Yampolskiy, Jinghui Toh, Jacob Gatlin, and Yuval Elovici. 2017. Dr0wned – {Cyber-Physical} Attack with Additive Manufacturing. https://www.usenix.org/conference/woot17/workshop-program/presentation/belikovetsky

[6] Simon Birnbach, Simon Eberz, and Ivan Martinovic. 2022. Haunted House: Physical Smart Home Event Verification in the Presence of Compromised Sensors. *ACM Trans. Internet Things* 3, 3 (2022), 18:1–18:28. doi:10.1145/3506859

[7] Avrim Blum and Tom Mitchell. 1998. Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory* (Madison, Wisconsin, USA) *(COLT' 98)*. Association for Computing Machinery, New York, NY, USA, 92–100. doi:10.1145/279943.279962

[8] Jeffery A. Brown, Leo Porter, and Dean M. Tullsen. 2011. Fast Thread Migration via Cache Working Set Prediction. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture* (2011-02). 193–204. doi:10.1109/HPCA.2011.5749728

[9] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2021. TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning. arXiv:2007.11622 [cs] doi:10.48550/arXiv.2007.11622

[10] Hsiang-Yun Sherry Chien, Javier S. Turek, Nicole Beckage, Vy A. Vo, Christopher J. Honey, and Ted L. Willke. 2021. *Slower Is Better: Revisiting the Forgetting Mechanism in LSTM for Slower Information Decay.* arXiv:2105.05944 [cs] doi:10.48550/arXiv.2105.05944

[11] Espressif Systems. 2025. *ESP-NOW - Arduino ESP32 Latest Documentation.* https://docs.espressif.com/projects/arduino-esp32/en/latest/api/espnow.html

[12] Yang Gao, Borui Li, Wei Wang, Wenyao Xu, Chi Zhou, and Zhanpeng Jin. 2018. Watching and Safeguarding Your 3D Printer: Online Process Monitoring Against Cyber-Physical Attacks. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 3 (2018), 108:1–108:27. doi:10.1145/3264918

[13] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Kurt Keutzer, Ion Stoica, and Joseph E. Gonzalez. 2020. *Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization.* arXiv:1910.02653 [cs] doi:10.48550/arXiv.1910.02653

[14] Yub Jung, Eun-Gyo Park, Seon-Ho Jeong, and Jeong-Ho Kim. 2024. AI-Based Anomaly Detection Techniques for Structural Fault Diagnosis Using Low-Sampling-Rate Vibration Data. *Aerospace* 11, 7 (2024), 509. Issue 7. doi:10.3390/aerospace11070509

[15] Rishi Kumar, Kuldip Singh Sangwan, Christoph Herrmann, Rishi Ghosh, and Mukund Sangwan. 2024. Development and Comparison of Machine-Learning Algorithms for Anomaly Detection in 3D Printing Using Vibration Data. *Progress in Additive Manufacturing* 9, 2 (2024), 529–541. doi:10.1007/s40964-023-00472-1

[16] Young D. Kwon, Jagmohan Chauhan, Hong Jia, Stylianos I. Venieris, and Cecilia Mascolo. 2023. *LifeLearner: Hardware-Aware Meta Continual Learning System for Embedded Computing Platforms.* arXiv:2311.11420 [cs] doi:10.48550/arXiv.2311.11420

[17] Ioannis Lamprou, Zhen Zhang, Javier de Juan, Hang Yang, Yongqiang Lai, Etienne Filhol, and Cedric Bastoul. 2023. Safe Optimized Static Memory Allocation for Parallel Deep Learning. *Proceedings of Machine Learning and Systems* 5 (2023), 305–324. https://proceedings.mlsys.org/paper_files/paper/2023/hash/676d8419c61f299feb88c28b40edd3b1-Abstract-mlsys2023.html

[18] Hunter Lang, Monica Agrawal, Yoon Kim, and David Sontag. 2022. Co-training Improves Prompt-based Learning for Large Language Models. arXiv:2202.00828 [cs.CL]

[19] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2024. *On-Device Training Under 256KB Memory.* arXiv:2206.15472 [cs] doi:10.48550/arXiv.2206.15472

[20] Zhiqing Luo, Mingxuan Yan, Wei Wang, and Qian Zhang. 2023. Non-Intrusive Anomaly Detection of Industrial Robot Operations by Exploiting Nonlinear Effect. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 4 (2023), 175:1–175:27. doi:10.1145/3569477

[21] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Ag"uera y Arcas. 2023. *Communication-Efficient Learning of Deep Networks from Decentralized Data.* arXiv:1602.05629 [cs] doi:10.48550/arXiv.1602.05629

[22] Johannes Obermaier and Stefan Tatschner. 2017. Shedding Too Much Light on a Microcontroller's Firmware Protection. https://www.usenix.org/conference/woot17/workshop-program/presentation/obermaier

[23] Colin O'Flynn and Alex Dewar. 2019. On-Device Power Analysis Across Hardware Security Domains.: Stop Hitting Yourself. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019), 126–153. doi:10.13154/tches.v2019.i4.126-153

[24] Valentin Panduru. 2025. Detection of 3-axis vibrations in faulty 3D printers using smartphones and machine learning. *Academic Journal of Manufacturing Engineering* 23, 1 (2025), L4. doi:10.5281/zenodo.15180494

[25] Shishir G. Patil, Paras Jain, Prabal Dutta, Ion Stoica, and Joseph E. Gonzalez. 2022. *POET: Training Neural Networks on Tiny Devices with Integrated Rematerialization and Paging.* arXiv:2207.07697 [cs] doi:10.48550/arXiv.2207.07697

[26] Christos Profentzas, Magnus Almgren, and Olaf Landsiedel. 2023. MiniLearn: On-Device Learning for Low-Power IoT Devices. In *Proceedings of the 2022 International Conference on Embedded Wireless Systems and Networks* (New York, NY, USA, 2023-01-18) *(EWSN '22)*. Association for Computing Machinery, 1–11.

[27] Ankit Singh Rawat, Priyank Anand, Hao Chen, and Pramod K. Varshney. 2011. Collaborative Spectrum Sensing in the Presence of Byzantine Attacks in Cognitive Radio Networks. *IEEE Transactions on Signal Processing* 59, 2 (2011), 774–786. doi:10.1109/TSP.2010.2091277

[28] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. 2021. TinyOL: TinyML with Online-Learning on Microcontrollers. In *2021 International Joint Conference on Neural Networks (IJCNN)* (2021-07). 1–8. doi:10.1109/IJCNN52387.2021.9533927

[29] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. 2021. 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)*. 126–135. doi:10.1145/3465480.3466928

[30] Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. 2023. A Comprehensive Survey on Model Quantization for Deep Neural Networks in Image Classification. *ACM Transactions on Intelligent Systems and Technology* (Sept. 2023). doi:10.1145/3623402

[31] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017-05). 195–212. doi:10.1109/SP.2017.14

[32] Abhik Sarkar, Frank Mueller, Harini Ramaprasad, and Sibin Mohan. 2009. Push-Assisted Migration of Real-Time Tasks in Multi-Core Processors. *SIGPLAN Not.* 44, 7 (2009), 80–89. doi:10.1145/1543136.1542464

[33] Zhangyue Shi, Abdullah Al Mamun, Chen Kan, Wenmeng Tian, and Chenang Liu. 2023. An LSTM-autoencoder Based Online Side Channel Monitoring Approach for Cyber-Physical Attack Detection in Additive Manufacturing. *Journal of Intelligent Manufacturing* 34, 4 (2023), 1815–1831. doi:10.1007/s10845-021-01879-9

[34] Sangeetha Siddegowda, Marios Fournarakis, Markus Nagel, Tijmen Blankevoort, Chirag Patel, and Abhijit Khobare. 2022. Neural Network Quantization with AI Model Efficiency Toolkit (AIMET). arXiv:2201.08442 [cs] doi:10.48550/arXiv.2201.08442

[35] Bharath Sudharsan, John G. Breslin, and Muhammad Intizar Ali. 2022. ML-MCU: A Framework to Train ML Classifiers on MCU-Based IoT Edge Devices. *IEEE Internet of Things Journal* 9, 16 (2022), 15007–15017. doi:10.1109/JIOT.2021.3098166

[36] Bharath Sudharsan, Pankesh Patel, John G. Breslin, and Muhammad Intizar Ali. 2021. Ultra-Fast Machine Learning Classifier Execution on IoT Devices without SRAM Consumption. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops)* (2021-03). 316–319. doi:10.1109/PerComWorkshops51409.2021.9431061

[37] Jun Xiao, Yixian Shen, and Andy D. Pimentel. 2022. Cache Interference-aware Task Partitioning for Non-preemptive Real-time Multi-core Systems. *ACM Trans. Embed. Comput. Syst.* 21, 3 (2022), 28:1–28:28. doi:10.1145/3487581

[38] Zhi-Hua Zhou. 2018. A Brief Introduction to Weakly Supervised Learning. *National Science Review* 5, 1 (Jan. 2018), 44–53. doi:10.1093/nsr/nwx106

[39] Zhi-Hua Zhou and Ming Li. 2005. Tri-training: exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering* 17, 11 (2005), 1529–1541. doi:10.1109/TKDE.2005.186

A Task Scheduling and Prioritisation Description

In Table 9, we explain the properties of each task running on the edge board using FreeRTOS. We implemented a custom task scheduler, as shown in Figure 6. The scheduler features summarised in Section 5.9 are realised through the per-task core assignments, priorities, triggers, and lock sets listed in Table 9. In our configuration, Core 0 handles the interrupt service routine (ISR) and the two highest-priority tasks (`tiny_fed`, `tiny_learn`), which manage FL exchanges and model updates. Core 1 executes `tiny_infer` and `tiny_colla` for low-latency AD and collaborative labelling. The

remaining tasks (`tiny_sense`, `tiny_proc`) have no fixed affinity, and the scheduler places them on the less-loaded core. This improves throughput and reduces latency for data collection and feature extraction. Some memory locations are accessed and modified by multiple tasks, creating critical sections where deadlocks may arise. Therefore, each task specifies the shared resources or buffers that require mutual-exclusion locks to maintain data consistency. For instance, if `tiny_learn` on Core 0 holds `windows_buf` and `tiny_infer` on Core 1 holds `inference_buf`, and each waits for the other's lock, a circular wait may result in a deadlock. Our task scheduling is designed to prevent this situation in the context of TinyLearnED.

Table 9. TinyLearnED Task Properties

Task	**Core**	**Pri.**	**Access**	**Function**	**Dependency**	**Trigger**	**Critical Sections (Locks)**
`tiny_fed`	PRO CPU (0)	3	weights; bias; timestamp	Manages federated-learning exchange of parameters and timestamps	`tiny_api`	Every R inferences ($R = 50$ or 150)	weights, bias
`tiny_learn`	PRO CPU (0)	3	weights; bias; windows buf.; inference buf.; label buf.	Fine-tunes the last layer of TinyFR	`tiny_colla`	Local prediction disagrees with pseudo-label	weights, bias; windows buf.; inference buf.; label buf.
`tiny_infer`	APP CPU (1)	3	weights; bias; windows buf.; inference buf.	Runs TinyFR inference on one batch; applies anomaly detection	`tiny_proc`	Every 3 windows (forms one batch)	weights, bias; windows buf.; inference buf.
`tiny_colla`	APP CPU (1)	3	inference buf.; ensemble buf.; label buf.	Exchanges peer predictions and computes ensemble decision	`tiny_api`, `tiny_infer`	When ensemble buffer is full	inference buf.; ensemble buf.; label buf.
`tiny_sense`	NO AFF. CPU	1	raw_windows buf.	Collects sensor signals and performs windowing	—	Periodic (50 Hz sampling)	raw_windows buf.
`tiny_proc`	NO AFF. CPU	2	raw_windows buf.; windows buf.	Preprocesses raw windows (DC removal, filtering, feature extraction)	`tiny_sense`	raw_windows buf. > 0 (new data)	raw_windows buf.; windows buf.
`tiny_api`	PRO CPU (0)	3	—	Establishes and manages WiFi/ESPNOW connections	—	System power-up	—