S^3OvA: A Reformable TinyML Solution for Self-Adaptive IoT-based Systems

Mohamed Maoui* and Rohallah Benaboud
Department of Mathematics and Computer Science, Research Laboratory on Computer Science's Complex Systems (ReLa(CS)2), University of Oum El Bouaghi, Oum El Bouaghi, Algeria
E-mail: mohamed.maoui@univ-oeb.dz, benaboud.rohallah@univ-oeb.dz
*Corresponding author

Keywords: Internet of things, self-adaptation, reformable tinyML, SVMs, SMO, incremental learning, OvA

Received: September 18, 2025

The proliferation of IoT-based systems necessitates intelligent edge devices capable of autonomously adapting to dynamic environments under highly constrained resource conditions. Yet current TinyML solutions rely predominantly on static models that cannot evolve after deployment, limiting their effectiveness in scenarios where environmental conditions and data distributions continuously change. To address this limitation, this paper presents S^3OvA (SVM, SMO, Syed with OvA), a reformable TinyML solution enabling progressive model updating directly on microcontrollers (MCUs) through hybrid offline/online methodologies. It integrates Support Vector Machines (SVM) with Sequential Minimal Optimisation (SMO) for efficient training, Syed et al.'s incremental learning method for dynamic updating, and the One-vs-All (OvA) strategy for multiclass extension, whilst implementing linear, polynomial, and RBF kernels with preserved computational efficiency on resource-constrained platforms. Experimental validation on the ESP32-S Module demonstrates accuracy improvements of up to 20 points through incremental learning alongside memory stability throughout adaptive cycles, whilst revealing empirically determined operational limits, and validating performance across binary classification, multiclass, and adaptive scenarios. Memory analysis further confirms implementation efficiency through stable heap usage and controlled resource progression. S^3OvA thus bridges the critical gap between static TinyML deployments and the dynamic requirements of IoT-based systems—reducing latency, preserving data privacy, and enhancing energy autonomy—establishing a new paradigm for self-adaptive edge computing wherein distributed intelligence emerges through continuous on-device learning, fundamentally transforming IoT-based systems from passive data collectors into autonomous decision-making entities.

Povzetek: Članek predstavi S^3OvA, prilagodljiv TinyML pristop za mikrokontrolerje, ki združi SVM+SMO, inkrementalno učenje in strategijo eden proti vsem, da omogoči sprotno posodabljanje modela na napravi ob stabilni porabi pomnilnika in brez odvisnosti od statičnih modelov.

1 Introduction

The term *Internet of Things (IoT)* was introduced over 25 years ago [1] and has since emerged as a transformative technological paradigm [2]. By interconnecting intelligent devices [3] within an *Intelligent IoT* framework—stemming from the convergence of *Cloud Computing*, *Fog Computing*, *Edge Computing*, and Artificial Intelligence (AI) [4]—IoT systems enable distributed data processing and decision-making. Typically powered by microcontrollers (MCUs) [5], these devices continuously collect environmental data and exchange it via the Internet with IoT applications, supporting real-time monitoring, optimisation, and service management [6]. This capability underpins autonomous operation across diverse domains, including home automation, healthcare, smart farming, and connected cities. However, IoT environments remain inherently uncertain due to evolving operational conditions [7], fluctuating resource availability, and shifting user objectives [8, 9], raising a central challenge: *how can IoT systems sustain real-time adaptation while preserving efficiency and relevance?*

To address this challenge, recent research has advanced Self-Adaptive Systems (SASs) leveraging Machine Learning (ML) techniques to dynamically adjust their behaviour [10], in line with Bures's vision [11] of *self-adaptation 2.0*—a convergence of AI and self-adaptation increasingly supported in the literature [12]. This adaptability has been investigated across multiple deployment paradigms. Cloud AI provides substantial computational capacity but entails high latency and network dependency [13]. Fog AI reduces these limitations by relocating computation closer to end devices [14, 15], although proximity networks such as Local Area Networks (LAN), Metropolitan Area Networks (MAN), and Wide Area Networks (WAN) may still impose latency and resource constraints [16]. Edge/Mobile AI executes processing directly on devices, enhancing privacy and energy efficiency [17, 18], yet hardware limitations hinder the deployment of complex ML models [19]. Model compression strategies—including *pruning*, *quantisation*, and *low-rank factorisation* [20]—together with specialised acceleration hardware [19, 21], have progressively enabled AI execution on resource-constrained platforms, ultimately leading to the emergence of *Tiny Machine Learning* (*TinyML*) [22].

TinyML refers to the execution of ML models directly on MCUs, enabling real-time applications with low latency and minimal power consumption across a wide range of IoT use cases [23]. However, TinyML models are generally *static*: pre-trained offline and unable to adapt to new data or environmental changes without manual intervention and retraining [24]. As data evolves, performance inevitably degrades—a phenomenon known as *model drift* [25, 26], often linked to *conceptual drift* [27]—highlighting the need for continuous adaptation. This has led to the concept of *reformable TinyML* [25], where integrated update mechanisms allow models to adjust their parameters in real time without full retraining. Rajapakse *et al.* [25] provide a detailed analysis of such solutions, introducing a taxonomy classifying them as *on-device offline learning*, *online learning*, and *network reliant approaches*.

Building upon these advancements, this paper introduces S^{3}OvA—a novel solution developed in the spirit of applied *online learning*, enabling ML models to be updated directly on MCUs while maximising performance in dynamic environments. This contribution converges three complementary advances: the adaptation of Support Vector Machines (SVMs) to MCUs via an optimised implementation of the Sequential Minimal Optimisation (SMO) algorithm [28] for binary classification; its extension to multiclass processing through the One-vs-All (OvA) strategy [29]; and the integration of an incremental learning method inspired by Syed *et al.* [30, 31] into this multiclass framework [32], enabling local model updates without full retraining.

These contributions were experimentally validated on a NodeMCU-32S_v1.3 board [33] integrating an ESP32-S Module [34] based on the ESP32-D0WD-V3 System-on-Chip (SoC) [35], whose heterogeneous memory hierarchy—comprising ROM (Read-Only Memory), internal SRAM (Static Random Access Memory), and 4 MB SPI (Serial Peripheral Interface) Flash memory—provided the computational substrate for local model execution and adaptation. Rigorous evaluation using synthetic benchmarks and real datasets confirms the viability of embedded incremental learning in dynamic IoT ecosystems and the strategic relevance of S^{3}OvA in advancing the TinyML paradigm.

The remainder of this article is structured as follows: Section 2 examines the landscape of reformable TinyML solutions; Section 3 introduces S^{3}OvA and its theoretical foundations; Section 4 details the implementation methodology; Section 5 presents the experimental evaluation; Section 6 synthesises the results and identifies optimisation pathways; Section 7 discusses the findings and outlines future research directions; and Section 8 concludes with broader implications for IoT research.

2 Related work

Reformable TinyML solutions rely on learning methods specifically optimised for the stringent constraints of IoT environments, including limited memory, tight energy budgets, and reduced computational capacity. This section analyses the principal approaches enabling such reformability, with particular attention to online, offline, and hybrid learning mechanisms that sustain continuous on-device model adaptation on MCUs. Among these, Benatti *et al.* [36] introduced PULP-HD, an ultra-low-power architecture leveraging hyperdimensional encoding on the parallel Mr. Wolf platform, enabling native online learning at only per inference—though still susceptible to catastrophic forgetting, partially mitigated through manual retraining. In a more memory-efficient direction, TinyTL [37] proposed updating only the bias parameters while freezing the main weights, yielding significant memory savings and a +32% performance boost, albeit at the cost of dependence on pre-trained models. Latent Replay [38] addressed forgetting by storing latent activations instead of raw data, combining this with partial freezing of deep layers to enhance stability, though requiring high-performance platforms like Snapdragon. Sudharsan *et al.* [39] presented Edge2Train, a `C++` framework enabling local training of binary SVMs on MCUs via a hybrid Online/Offline pipeline that integrates real-time inference, periodic SMO-based retraining relying on full model re-optimisation, closed-loop feedback, and a *Data Stitching* mechanism to generate training sets autonomously—thereby ensuring energy efficiency, cloud independence, and MCU compatibility. Disabato and Roveri [40] followed a similar hybrid design, combining a pre-trained CNN (Convolutional Neural Network) with an embedded incremental k-NN (k-Nearest Neighbours) classifier on STM32, balancing accuracy and memory constraints, though scalability remains a concern. Lightweight alternatives include the online dataset adaptation method of Lee and Nirjon [41], which relies on per-output feature distributions and weight-based feature significance to enable efficient on-device adaptation without explicit backpropagation, but remains effective mainly under relatively stable conditions. De Prado *et al.* [42] proposed a hybrid offline/online imitation learning system using quantisation and GAP8 acceleration, demonstrating strong results for autonomous miniature vehicles, albeit with off-device retraining requirements. In fully online paradigms, TinyOL [43] offers a simple yet robust design involving a frozen model and SGD-updated output layer, validated on Arduino Nano and extended through variants (TinyOL v2, CWR+, LWF) by Avi *et al.* [44] for industrial scenarios. This line of work culminated in 2024 [45] with an integrated pipeline combining TinyOL, TinyMetaFed (federated meta-learning), and SeLoC-ML (semantic adaptation), achieving notable gains (+18% accuracy, –32% energy consumption). Complementary efforts include Train++ [46] for sample-wise updates on low-cost MCUs, Imbal-OL [47] for adaptive handling of class imbalance through dynamic oversampling and weighting, and QLR-CL [24] which couples latent replay with quantisation to mitigate forgetting—albeit with higher computational cost. For binary classification, ML-MCU [48] leverages a One-vs-One (OvO) strategy yielding +34.1% accuracy over TinyTL, though unsuitable for ultra-constrained MCUs. Notably, TyBox [49] provides a fully integrated on-device incremental learning solution, validated across diverse datasets and directly deployable on commercial MCUs. More recently, Rüb *et al.* [50] introduced a hybrid approach combining offline distillation and embedded incremental adaptation, enabling structurally flexible and resilient reformable architectures for future autonomous IoT systems.

3 S^{3}OvA: proposed solution

Although several studies have explored online learning methods based on compact neural networks or partial update strategies, few have fully leveraged SVMs in embedded systems. Yet, SVMs stand out for their robustness, accuracy, and efficiency, including when trained on small samples [51]. This efficiency is largely attributed to their remarkable ability to generalise from a limited subset of relevant examples—the support vectors—[52], thereby enabling a combination of memory efficiency, low training complexity, and resilience to noise or data scarcity. Moreover, several works have demonstrated their capacity to track evolving online data streams by dynamically readjusting decision boundaries, making them particularly effective for handling concept drift in non-stationary environments [31, 52, 53].

From these observations, the S^{3}OvA solution builds upon fundamental SVMs properties while incorporating complementary mechanisms to enable: incremental model updating, extension to multiclass classification, and efficient implementation in resource-constrained embedded environments. It judiciously combines: kernel methods for non-linear processing, the incremental learning method by Syed *et al.* [30] for model updates, local optimisation via SMO, and multiclass structuring through OvA strategy—collectively forming a unified theoretical and practical framework whose foundational principles, design methodology and implementation will be systematically elaborated.

3.1 Foundational principles

S^{3}OvA is primarily grounded in the theoretical foundations of SVMs, originally introduced by Vladimir Vapnik within the framework of statistical learning theory [54]. Initially developed for linear binary classification tasks, SVMs aim to identify an optimal hyperplane of the form $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ which separates data from two distinct classes with the maximum possible margin, while simultaneously minimising the risk of classification error.

This problem can be formalised as:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to:} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1,\ \forall i \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ are the input vectors, $y_i \in \{-1, +1\}$ the class labels, $\mathbf{w}$ the normal vector to the hyperplane, and b the bias term.

This primal formulation corresponds to the linear and perfectly separable case. In practice, data is often not perfectly separable, leading to the introduction of slack variables ξ_i and a regularisation parameter C, resulting in the soft-margin SVM formulation:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to:} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \end{aligned} \tag{2}$$

This model is subsequently reformulated in its dual form to facilitate its extension to non-linear cases.

The dual formulation is expressed as follows:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x_i}, \mathbf{x_j}) \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{3}$$

Indeed, beyond a basic linear formulation, S^{3}OvA accommodates non-linear scenarios through the implementation of kernel functions, which enable the implicit projection of data into a higher-dimensional feature space without explicitly computing the transformation. Accordingly, the decision function becomes:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b \tag{4}$$

where $\kappa(\cdot,\cdot)$ is a kernel function (e.g., linear, Radial Basis Function (RBF), polynomial), and α_i are the Lagrange multipliers optimised during training.

This ability to model complex decision boundaries while preserving an efficient formulation has enabled S^{3}OvA to establish itself across various application domains, including those requiring high accuracy on limited data volumes. However, in dynamic environments characterised by continuous and non-stationary data streams, a structural limitation emerges: standard SVMs rely on the global resolution of a convex optimisation problem, which makes their incremental adaptation non-trivial. Indeed, the inclusion of new data may invalidate the previously computed optimal solution, as it depends on the entirety of the available dataset.

To address this limitation, Syed *et al.* proposed an incremental learning method in which the training problem is reformulated in the dual space. In this framework, the optimal solution depends solely on the support vectors—those critical points that define the separating hyperplane. The core idea is to treat the arrival of a new batch of samples

$$\mathcal{F}_{t+1} = \left\{(\mathbf{x}_1^{t+1}, y_1^{t+1}), \ldots, (\mathbf{x}_N^{t+1}, y_N^{t+1})\right\}$$

as a local perturbation, and to update only the α_i associated with the portion of the model affected by this change.

Within this perspective, each new sample is assessed according to its potential impact on the current solution. If it alters the decision space, a local update of α_i is triggered by solving a reduced subproblem. This methode—illustrated in Figure 1 and detailed in Algorithm 1—offers several advantages: (i) amortised near-constant complexity in favourable conditions, (ii) preservation of marginal support vectors S_t^m and bounded ones S_t^b, while discarding non-informative points for which $\alpha_i = 0$, and (iii) continuous adaptation to concept drift without requiring exhaustive data storage.

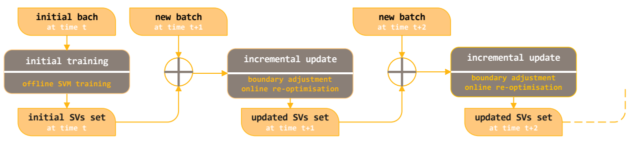


Figure 1: Workflow of Syed *et al.*'s incremental learning method.

Algorithm 1: Syed *et al.*'s incremental learning method.

```
Require: $\mathcal{F}_{t+1} = \{(\mathbf{x}_1^{t+1}, y_1^{t+1}), \ldots, (\mathbf{x}_N^{t+1}, y_N^{t+1})\}$: new samples at time $t+1$
         $\mathcal{S}_t^m$: margin support vector set at $t$
         $\mathcal{S}_t^b$: bounded support vector set at $t$
         $f_t$: decision function at $t$
Ensure : $f_{t+1}$: updated decision function
         $\mathcal{S}_{t+1}^m$: margin support vector set at $t+1$
         $\mathcal{S}_{t+1}^b$: bounded support vector set at $t+1$
1 Begin
2     Compute $\tilde{\mathcal{F}}_{t+1} = \mathcal{F}_{t+1} \cup \mathcal{S}_t^m \cup \mathcal{S}_t^b$
3     Using $\tilde{\mathcal{F}}_{t+1}$ as input, solve the problem of Equation (3) to
      obtain $\alpha_i^{t+1}$ for all $i$
4     Compute $f_{t+1}$ by solving Equation (4)
5     Compute
      $\mathcal{S}_{t+1}^m = \mathcal{S}_t^m \cup \{\mathbf{x}_i^{t+1} : y_i^{t+1} f_{t+1}(\mathbf{x}_i^{t+1}) = 1 \wedge \alpha_i^{t+1} > 0\}$
6     Compute
      $\mathcal{S}_{t+1}^b = \mathcal{S}_t^b \cup \{\mathbf{x}_i^{t+1} : y_i^{t+1} f_{t+1}(\mathbf{x}_i^{t+1}) < 1 \wedge \alpha_i^{t+1} = C\}$
7     Discard all non-informative points for which $\alpha_i^{t+1} = 0$
8     return $f_{t+1}$, $\mathcal{S}_{t+1}^m$, and $\mathcal{S}_{t+1}^b$
9 End
```

However, despite these benefits, this incremental methode relies on an iterative resolution of the dual optimisation problem (see Equation (3)), the computational cost of which may quickly become prohibitive in embedded settings with limited resources.

Confronted with this constraint, a particularly well-suited optimisation algorithm stands out: SMO. This method offers an efficient solution to the SVM dual problem by circumventing costly global matrix operations. More specifically, it decomposes the quadratic optimisation task into a sequence of two-variable subproblems, each of which is solved analytically. This decomposition significantly reduces computational complexity while preserving the model's accuracy.

The SMO algorithm, whose overall pseudo-code is presented in Section 2.5 of [28] and Section 12.3 of [55], proceeds by selecting at each iteration a pair of Lagrange multipliers, α_i and α_j, which it locally optimises under the constraint of zero-sum variation. The decision function is then updated incrementally, without the need for a full re-optimisation.

Accordingly, S^{3}OvA can effectively adapt to dynamic and resource-constrained environments, as typically encountered in embedded systems. However, this capability relies on a binary classification formalism. Yet, many embedded applications require multiclass decision-making. To meet this requirement, the S^{3}OvA solution adopts a canonical extension of SVMs to the multiclass setting, based on the OvA strategy. This approach involves training K independent binary classifiers, each responsible for distinguishing a target class k from all remaining classes. The final prediction is given by:

$$\hat{y} = \arg\max_{k \in \mathcal{Y}} f^{(k)}(\mathbf{x})$$

where $f^{(k)}$ denotes the decision function of the classifier associated with class k, and $\mathcal{Y} = \{1, \ldots, K\}$ is the set of possible classes.

Each binary task is defined via the following label transformation:

$$y_i^{(k)} = \begin{cases} +1, & \text{if } y_i = k \\ -1, & \text{otherwise} \end{cases}$$

This modular and inherently parallelisable structure constitutes the foundation of S^{3}OvA. It allows for a coherent integration of the different components—incremental learning, margin adjustment, and multiclass extension—following a clear separation-of-concerns principle, which is conducive to efficient implementation within embedded systems.

3.2 Design methodology

S^{3}OvA's modular structure is a key feature of its design, directly derived from the principle of *separation of concerns*, which isolates responsibilities related to initial model training from those related to dynamic updating whilst ensuring their consistent interaction. This principle is rooted in Parnas's pioneering work [56], whereby a module encapsulates a single design decision to limit interdependencies [57] and facilitate system evolution [58]. In this paper, modularity is understood as a logical and/or functional grouping—whether code blocks, classes, or functions—that can be reused, tested, and updated independently [59], in line with the recommendations of Sarkar *et al.* [60], for whom increased modularity promotes long-term maintainability and extensibility through metrics compatible with SOLID principles [61]. Accordingly, S^3OvA's design methodology is based on a *modular architecture with two complementary phases*—an offline phase dedicated to initial learning, and an online phase geared towards dynamic updating—configurably integrating the SMO algorithm and SVM kernels as the default configuration, thereby promoting flexibility, progressive adaptation to real data flows, and compatibility with resource-constrained embedded deployments [58].

3.2.1 Offline phase—initial training

This phase constitutes the preliminary stage of model construction, carried out on a static and complete dataset previously binarised via the OvA strategy to decompose the multiclass problem into independent binary sub-problems (see Figure 2). Two training modes can be distinguished. *Batch Training* trains the model conventionally on the entire binarised dataset (see Figures 2(a) and 2(c)), producing a high-performance reference model by maximising data usage. *Simulated Online Learning (Mini-Batch Streaming)*, by contrast, artificially segments the training set into stratified mini-batches to evaluate the incremental learning method under controlled conditions (see Figures 2(b) and 2(c)): at each iteration, the model is locally updated by integrating new samples whilst retaining the relevant support vectors. At the end of this phase, the trained models—one per class under the OvA strategy—are stored for subsequent dynamic updates.

3.2.2 Online phase—dynamic update

The online phase enables S^3OvA to handle a continuous stream of new samples in real time within a non-stationary environment, automatically orchestrating three complementary actions. Each incoming sample is first subjected to *Prediction*, whereby its class $\hat{y}_i$ is estimated using the current set of binary classifiers (see Figures 3(a) and 2(c)). Should the sample prove informative—i.e., altering the decision boundary—an *Update* is triggered, locally adjusting the corresponding α_i following the incremental method inspired by Syed *et al.* (see Figures 3(b) and 2(c)). Finally, upon detection of an unknown class, *Model Extension* seamlessly incorporates a new binary classifier alongside the existing ones, progressively enriching the structure without disruption (see Figures 3(c) and 2(c)). This process ensures S^3OvA's continuous adaptability to contextual changes—such as concept drift or the emergence of new classes—without requiring full retraining, making it particularly well suited to the constraints of intelligent and distributed embedded systems.

4 Implementation

The architectural modularity of S^3OvA represents a key asset for its implementation on MCUs, offering a structure capable of self-adaptation to data evolution while meeting the stringent constraints inherent to such devices. MCUs are characterised by limited memory, reduced computational power, the frequent absence of a full-fledged operating system, and stringent requirements in terms of robustness, reliability, and energy efficiency. Moving from theoretical design to practical implementation therefore requires a coherent and integrated set of design principles that address two complementary dimensions: software and hardware.

From a software perspective, the foremost requirement is maximum portability across hardware targets without Operating System (OS) dependencies, enabling deployment on MCUs with or without an Real-Time Operating System (RTOS). Dynamic memory allocation is managed in accordance with established standards, in order to prevent memory leaks, control fragmentation, and ensure deterministic behaviour in real-time environments. Numerical precision is grounded in `float32`, which strikes a practical compromise between SMO convergence stability and memory footprint (4 B per value). The implementation is written in `C++`, selected for its efficiency, low-level control, and native support across embedded platforms, with modern `C++11` practices adopted to enhance readability, memory safety, and maintainability [62]. Reliance on external libraries is deliberately minimised, with critical components implemented in-house to tighten resource control and eliminate unnecessary overhead. At the algorithmic level, careful tuning of SMO hyperparameters—including `tol`, `maxIter`, and `alphaTol`—ensures fast convergence and numerical stability in an incremental learning setting. Frequent kernel value accesses are addressed through an intelligent caching strategy balancing memory consumption and computational performance, whilst RBF and polynomial kernels are encapsulated as independent modules with explicit attention to their computational cost and cache compatibility.

From a hardware perspective, a 32-bit CPU provides an optimal balance between complexity, energy consumption, and memory addressing, offering a sufficiently robust foundation whilst remaining compatible with low-cost embedded systems. The integration of a Floating Point Unit (FPU) is essential to accelerate kernel computations and the resolution of the SMO's Quadratic Programming problem, avoiding the overhead of software emulation. Finally, S^3OvA's viability depends on a well-balanced memory architecture combining volatile SRAM and persistent Flash storage—a fundamental requirement for ensuring both processing efficiency and scalability with respect to the number of classes.

5 Experimental evaluation

5.1 Experimental configuration

5.1.1 Selection of the software and hardware platform

At this stage of development, the selection of the experimental platform is driven by the need to ensure close alignment with the software and hardware constraints previously defined. In this context, the ESP32-S Module, which integrates the ESP32-D0WD-V3 SoC, was selected. This processor features a dual-core, 32-bit Xtensa LX6 CPU running at 240 MHz, with a built-in FPU, meeting the floating-point computational requirements imposed by S^3OvA.

The ESP32-D0WD-V3 SoC provides 520 kB of SRAM, part of which is allocated to the RTC domain, enabling the efficient loading of temporary internal structures such as support vectors, update buffers, or kernel matrices. An external 4 MB SPI Flash memory (provided by the ESP32-S Module, since the CPU itself does not include embedded flash) ensures persistent storage of binary classifiers, optimised hyperparameters, and training datasets. This configuration meets the modular memory constraint necessary to support both phases of the S^3OvA pipeline.

On the software side, the selected environment is ESP-IDF (Espressif IoT Development Framework) [63], due to its direct compatibility with the hardware architecture and its ability to operate without an RTOS. This enables fine-grained low-level control over memory and CPU resources, ensuring full compliance with the constraint of a bare-metal or minimally orchestrated execution model.

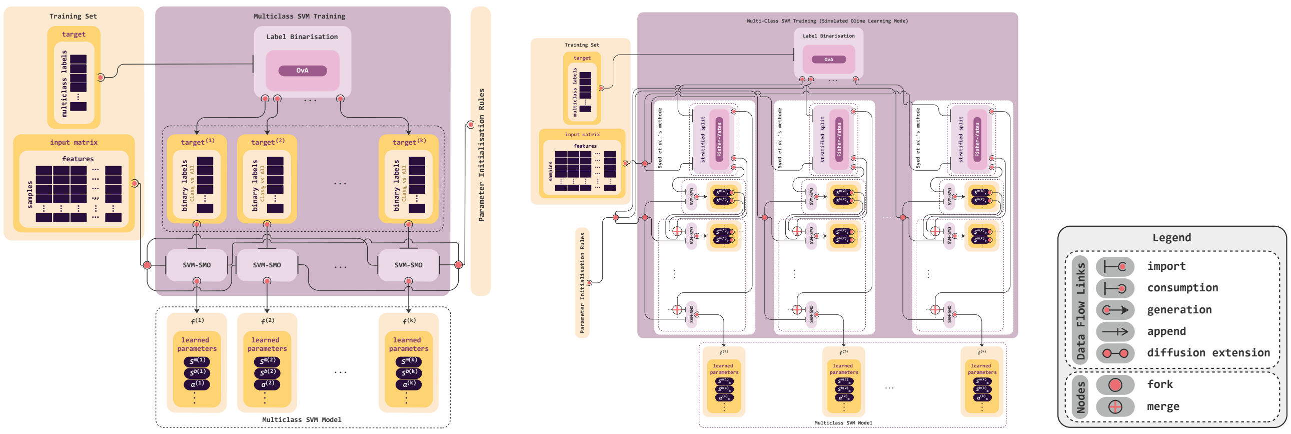
5.1.2 Progressive validation strategy

The experimental validation of S^3OvA follows a structured incremental strategy comprising three stages, motivated by the memory and computational constraints of the ESP32-S Module as well as its modular design. The first stage validates the functional core through independent binary classifiers, evaluated on both synthetic and real datasets encompassing linear and non-linear cases; classification accuracy, training time, inference latency, and memory footprint are assessed to establish a performance baseline prior to any extension. Building on these observations, the second stage estimates SRAM and Flash requirements to determine the maximum usable dataset size, then deploys batch training to pre-train the OvA binary classifiers before transitioning to simulated online learning mode (see Figure 2), thereby evaluating S^3OvA's scalability with respect to the number of classes as well as the stability and compactness of the embedded models. The third stage finally tests full evolution scenarios—including prediction on new samples, model updates, and extension to previously unseen classes—in order to validate S^3OvA's capacity for incremental self-adaptation without compromising overall performance, with metrics targeting robustness, temporal adaptability, and dynamic memory footprint.

5.2 Binary case: validation of the S^3OvA core

5.2.1 Selection of datasets

This phase relies on a dual-strategy combining controlled synthetic datasets and representative real-world data, aiming to balance fine-grained analysis of S^3OvA's internal behaviour with validation in scenarios approximating practical applications, whilst respecting the computational constraints of the ESP32-S Module. On the synthetic side, 27 datasets were generated using `make_classification` from `sklearn.datasets` [64], enabling precise control over structural parameters and isolated evaluation of their impact on embedded performance. Generation was parameterised along two dimensions: increasing



(a) Batch training mode. (b) Simulated online learning mode. (c) Legend.

Figure 2: S^3OVA solution: offline phase

dimensionality according to the first nine terms of the Fibonacci sequence, ensuring gradual and representative scaling, and three sample sizes per dimension (100, 150, and 200 examples), yielding 27 distinct training sets. Additional parameters govern the number of informative versus redundant features, class and cluster configurations, stochastic label noise, hypercube-based spatial distribution, shuffling for non-linear separability, and a fixed random seed ensuring reproducibility.

On the real-world side, five well-established benchmark datasets were selected—*Haberman* [65], *Banknote* [66], *Adult* [67], *Breast* [68], and *Spam* [69]—each sub-sampled using the same three-size pattern, producing 15 real-world training sets. Most are class-balanced, facilitating comparative analysis; the Haberman 200-sample subset, which exhibits notable class imbalance, was intentionally preserved and subsequently corrected via Synthetic Minority Over-sampling Technique (SMOTE) [70] to assess S^{3}OvA's robustness to distributional bias.

5.2.2 Deployment of the S^{3}OvA core

The integration of S^{3}OvA on the ESP32-S Module (see Figures 4 and 2(c)) relies on a modular C++ implementation tailored to its execution and memory constraints, facilitating maintainability, functional extension, portability, and deterministic execution. The software architecture is organised around four dedicated header files: `SVM-SMO.h` encapsulates the main data structures, initialisation functions, and the SMO-based optimisation loop adapted to the module's computational limits; `kernels.h` groups the kernel functions, dynamically invoked according to model configuration for flexible and reusable kernel management; `data.h` centralises training vectors, labels, and intermediate structures (α_i, support vectors, errors); and `S3OvACore.ino` serves as the main entry point, orchestrating data loading, parameter initialisation, optimisation calls, and incremental model updates.

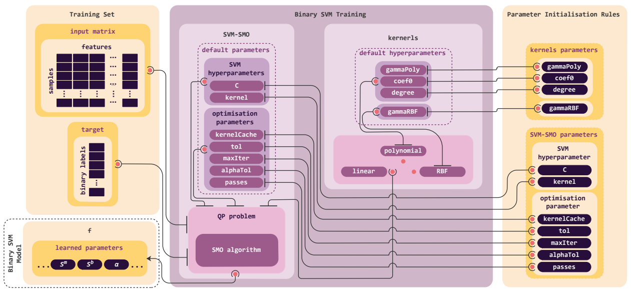
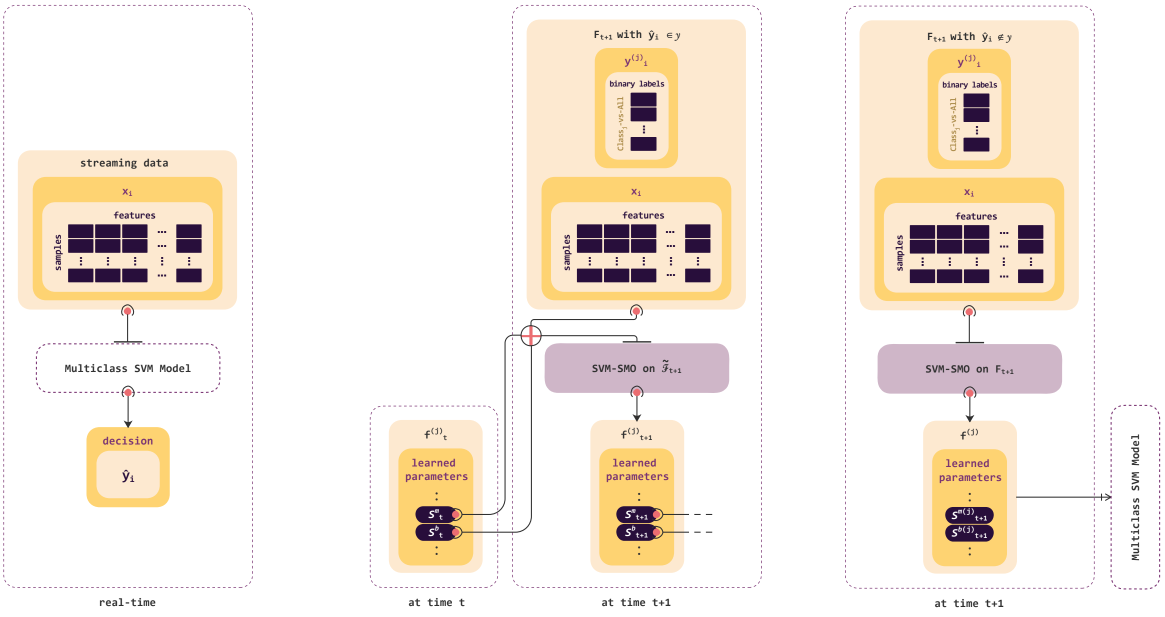


Figure 4: S^{3}OvA core

The project was developed and deployed using Arduino IDE 2 with the ESP-IDF compiler, ensuring full compatibility with the ESP32-D0WD-V3 SoC and enabling seamless cross-compilation, step-by-step serial debugging, and optimised firmware uploading via native ESP32 bootloader integration. From a memory standpoint, constants and static vectors are placed in Flash using the `const` qualifier, whilst computation buffers are dynamically allocated in SRAM with continuous heap monitoring to prevent overflows and fragmentation; Flash-to-SRAM transfers are handled via `memcpy_P` for improved efficiency. Training and inference durations are tracked through `micros()` and `millis()` timestamping, enabling precise identification of performance bottlenecks, particularly during non-linear kernel computations within



(a) Prediction process. (b) Incremental update process. (c) Model extension process.

Figure 3: S^{3}OVA solution: online phase

the optimisation phase.

5.2.3 Results and comparative analysis

Results on synthetic and real-world datasets On synthetic datasets, the comparative evaluation reveals significant disparities in classification performance, computational efficiency, and memory usage, with overall remarkable results (see Figure 5). The linear kernel consistently outperforms others, achieving accuracy exceeding 92% in most configurations and peaking at 97.50% for higher feature dimensions (Number of Features (NF) $\in \{21, 34, 55\}$). Nonetheless, specific configurations ($(\text{Number of Samples (NS)}, \text{NF}) \in \{(100,5), (150,3), (200,2), (200,3)\}$) constitute notable exceptions, where higher-degree polynomial kernels and the RBF kernel ($\gamma = 0.22$) outperform the linear model, suggesting latent non-linear structures requiring more sophisticated feature space transformations. Confusion matrix analysis corroborates these observations: failing configurations exhibit balanced false positive/negative distributions indicative of inadequate decision boundaries, whilst high-performing ones show minimal error incidence. Beyond classification accuracy, computational efficiency follows a similarly nuanced pattern: the linear kernel demonstrates optimal training-time efficiency in most cases, except for NF $\in \{8, 13\}$ where a noticeable overhead is observed, whilst prediction times show greater divergence, with RBF kernels requiring 3 to 4 times longer than polynomial kernels in high-dimensional settings. Regarding memory usage, distinct patterns emerge across kernels: degree-3 polynomial kernels record the highest Programme Storage Space (PSS) consumption—reaching 332 973 B for NS = 200 and NF = 55 (25.40% of available memory)—whilst dynamic memory grows moderately with data complexity. Consequently, heap cache monitoring shows stable behaviour up to NS = 150, but necessitates activation of SMO cache optimisation (`setUseCache`) at NS = 200, preserving model quality whilst maintaining system stability.

Turning to real-world datasets, the evaluation reveals generally satisfactory yet more variable performance (see Figure 6). The Haberman dataset, characterised by its inherent difficulty, yields relatively low accuracy with small sample sizes—45%–60% for 100 samples and 50%–63.33% for 150, with the degree-3 polynomial kernel ($b = -0.4$) achieving the best result (60%) whilst the RBF kernel ($\gamma = 0.22$) caps at 45%, equivalent to random prediction. Addressing the class imbalance at NS = 200 through SMOTE-based rebalancing (100 survivors / 100 deaths) substantially improves performance: 72.50% for the degree-3 polynomial, 70% for both linear and RBF kernels, and 67.50% for the degree-2 polynomial. In contrast, the Banknote dataset yields particularly remarkable results, reaching 100% accuracy and demonstrating the models' effectiveness on structurally well-defined data. The Adult dataset exhibits size-dependent fluctuations (65%–100% at NS = 100, 80%–96.67% at NS = 150, 75%–90% at NS = 200), whilst the higher-dimensional Breast Cancer (30 features) and Spam (57 features) datasets consistently exceed 85%, with Spam reaching 85% at NS = 100 and 90% at NS = 150, the best results being obtained with polynomial kernels. Across all datasets, the γ parameter of the RBF kernel proves complex and dataset-dependent, requiring case-specific optimisation, whilst degree-3 polynomial models consistently offer the best runtime-accuracy trade-off compared to RBF models ($\gamma = 0.22$), which incur significantly longer inference durations. Memory behaviour mirrors the patterns observed in synthetic tests, with resource impacts remaining within acceptable limits—until a critical constraint emerges during Spam testing at NS = 200 and NF = 57: a memory allocation error caused by a 34 224 B overflow of the `dram0_0_seg` segment prevents compilation, exposing the physical limits of the ESP32-S Module when handling high-dimensional models.

Comparative analysis The comparative analysis conducted around S^{3}OvA core highlights significant advancements in the field of reconfigurable TinyML solutions with offline on-device learning. Based on a rigorous evaluation using the *Iris* [71] and *MNIST* [72] benchmarks, the results establish a new reference point for the trade-off between performance and resource consumption compared to existing state-of-the-art approaches (see Table 1; hardware details in Tables 2 and 3), whilst adhering to the strict constraints of resource-limited embedded environments.

The most notable contribution resides in training performance. On the Iris dataset, S^{3}OvA core achieves training in 90 ms with a perfect accuracy of 100%, yielding a 113.2× speed-up over the ESP32-WROOM-32 Module (10 187 ms) and a 100.4× improvement over the Generic ESP32 board (9037 ms), whilst maintaining a precision gain of 3.33 to 13.33 points. This advantage is further confirmed on MNIST, where speed-up factors ranging from 8.7× to 11.7× accompany accuracy gains of 0.01 to 8.33 points, underscoring algorithmic robustness in the face of increasing complexity. Beyond raw performance, a key advantage lies in cross-platform consistency: in contrast to Edge2Train, whose results vary significantly across hardware, S^{3}OvA core

[1] Kaggle-based virtualised environment: `https://www.kaggle.com`.

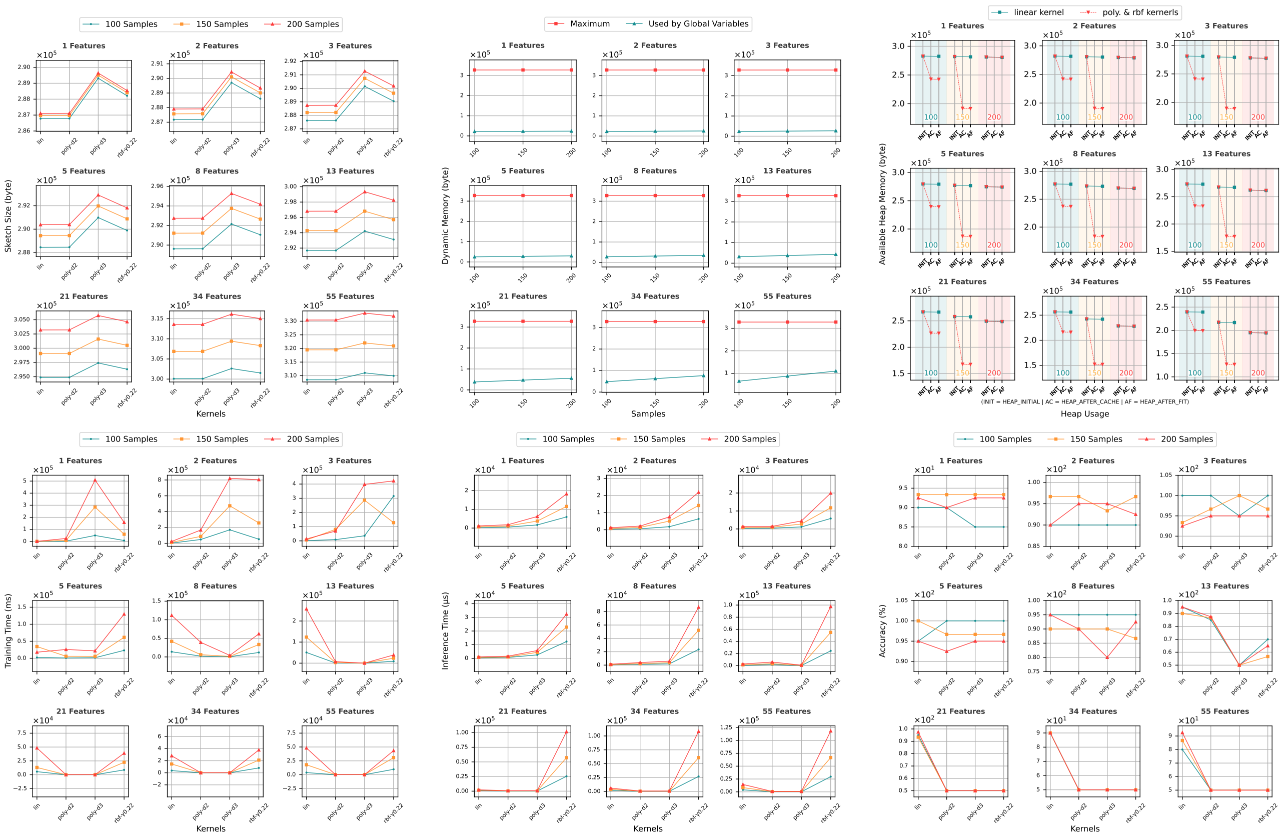


Figure 5: S^{3}OvA core: results on synthetic datasets.

maintains stable accuracy—100% on Iris and 91.67% on MNIST—across both embedded and laptop CPUs, confirming its portability and algorithmic stability. Comparisons with desktop-grade `scikit-learn` (backed by `LIBSVM` [75], implementing the optimised SMO decomposition of Fan *et al.* [73] for `SVC` [76]) further reflect the algorithmic maturity of S^{3}OvA core: whilst `scikit-learn` retains a marginal training-time advantage on laptops, S^{3}OvA core delivers equivalent classification accuracy whilst remaining natively executable on a MCU, thus eliminating any compromise between classification performance and embedded integrability. From a resource standpoint, full training is completed in under one second for Iris and under eight seconds for MNIST, opening new possibilities for low-latency local learning and suggesting promising applicability to more complex real-world problems, as evidenced by the empirical scalability observed between Iris (4 features) and MNIST (64 features).

That said, inference times remain higher than those of Edge2Train—295 µs versus 31 µs to 34 µs on Iris, and 4010 µs versus 100 µs on MNIST—reflecting a deliberate design choice that prioritises training quality over prediction speed, a trade-off that proves pertinent in scenarios where decision accuracy takes precedence over ultra-low-latency requirements, particularly when reduced training times enable more frequent model updates. Taken together, these results empirically validate the robustness, portability, and practical relevance of S^{3}OvA core, representing a decisive step towards more reactive, accurate, and self-sufficient embedded intelligence aligned with the growing demands of adaptive IoT.

5.3 Multiclass case: scalable extension

5.3.1 Preliminary optimisation

Technical Insights from the Binary Case The in-depth evaluation of S^{3}OvA core has revealed several structural limitations that must be anticipated prior to any extension towards a multiclass classification framework—constraints that should be viewed not merely as impediments, but as essential anchor points for optimising adaptability and robustness in more complex scenarios. Flash memory analysis exposes a primary concern: of the 1.25 MiB allocated to user code, S^{3}OvA core consumes up to 25.40% under a standard configuration (NS = 200, NF = 55). Whilst this usage remains controlled in the binary case, it will inevitably increase with the number of samples, which itself scales proportionally with the number of classes in balanced multiclass configurations—posing a clear risk of resource saturation on embedded platforms.

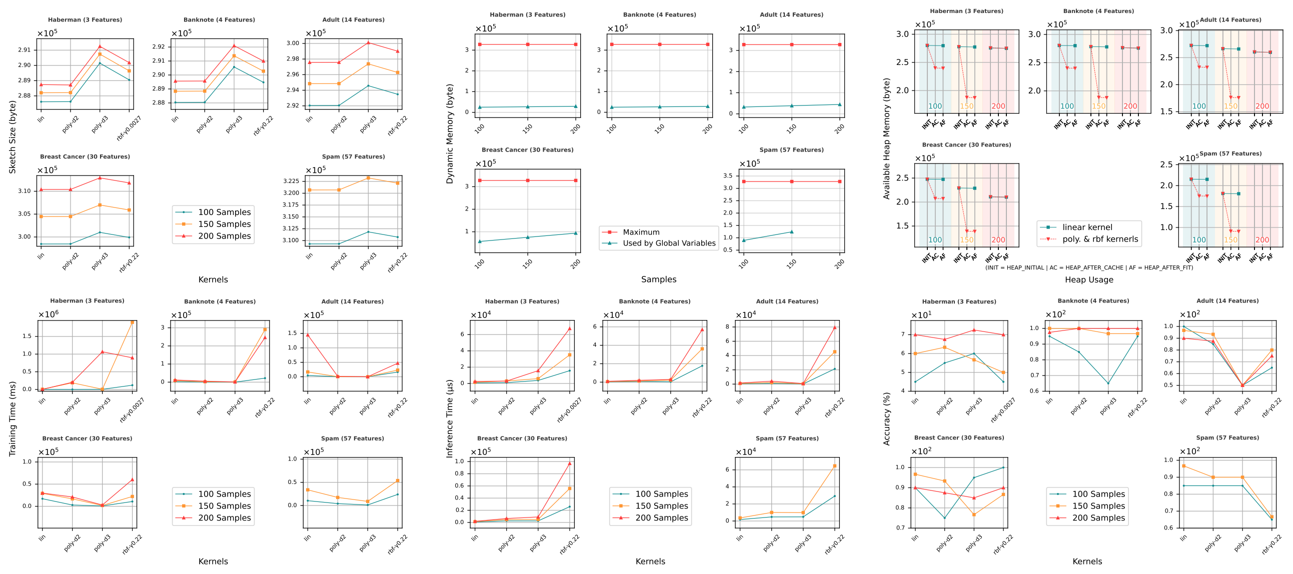


Figure 6: S^3OvA core: results on real-world dataset.

Table 1: Comparative performance results of the S^3OvA core.

DN	NF	NS	Module	Class	#	DM	Param. Init. Rules				Execution Output		
							C	κ	tol	maxIter	TT	IT	ACC
Iris	4	100	S^3OvA core	SVM-SMO	S1-DB1	[28, 55]	1	Linear	1e-5	10000	90	295	100.00
					S1-M1						194.427	646	100.00
			ToD Block	SVMSMO	S2-DB1						10187.0	31	86.67
					S2-DB2						9037.0	34	90.0
					S2-M1						1.193	6	93.37
					S2-M2						1.738	4	96.67
			svm	SVC	S3-M1	[73]					1.374	978	100.00
MNIST	64 (PCA Dim. Red.)	120	S^3OvA core	SVM-SMO	S1-DB1	[28, 55]	1	Linear	1e-5	10000	2168	4010	91.67
					S1-M1						838.241	1956	91.67
			ToD Block	SVMSMO	S2-DB1						25326.0	100	86.12
					S2-DB2						18754.0	100	83.34
					S2-M1						2.516	341	91.66
					S2-M2						2.671	366	88.89
			svm	SVC	S3-M1	[73]					2.018	661	91.67

Abbreviations: DN: Dataset Name; NF: Number of Features; NS: Number of Samples; #: Solution-Development Board/Machine; DM: Decomposition Method; Param. Init. Rules: Parameter Initialisation Rules; C: Regularisation Parameter; κ: Kernel Type; tol: Tolerance; maxIter: Maximum Iterations; TT: Training Time (ms); IT: Inference Time (µs); ACC: Accuracy (%); PCA Dim. Red.: Principal Component Analysis Dimensionality Reduction.

This concern naturally extends to the management of memory allocated to global variables, which follows a predictable linear behaviour, modelled by the following analytical equation:

$$M_{GVs}(NS, NF) = M_{GVs}^{init} + \left(\frac{NS - 100}{50}\right) \cdot (800 + (NF - 1) \cdot 400) + (NF - 1) \cdot 800 \quad (5)$$

where M_{GVs}^{init} denotes the baseline consumption for the configuration (100, 1).

This model enables memory usage forecasting as a function of dataset size, although deviations observed on real-world datasets such as Haberman, Breast, and Spam suggest the presence of context-specific compiler optimisations that must be considered for precise resource sizing in each use case. Beyond these foreseeable constraints, a major hardware limitation has been identified concerning the `dram0_0_seg` memory segment, used for fast-access data: an overrun of approximately 34.5 kB observed on the Spam dataset (NS = 200, NF = 57) resulted in a compilation error, highlighting this segment—whose usable capacity often remains below 160 kB [63] (see Chap. 4, Sec. 4.23)—as a hard ceiling directly conditioning the feasibility of embedded multiclass training. This limitation is further reflected in the evolution of the initial free heap memory, which exhibits a regular pattern modelled by the empirical formula:

Table 2: Development boards used for evaluation.

	Development Board							
Solution	#	Name	Module	SoC (Chip)	CPU	FPU	SRAM	Flash
S^{3}OvA	S1-DB1	NodeMCU-32S_v1.3	ESP32-S	ESP32-D0WD-V3	Xtensa® dual-core 32-bit LX6 240 MHz	Yes	520 kB	4 MB
Edge2Train	S2-DB1	Adafruit HUZZAH32	ESP32-WROOM-32	ESP32-D0WDQ6	Xtensa® dual-core 32-bit LX6 240 MHz	Yes	520 kB	4 MB
	S2-DB2	Generic ESP32	NA	NA				

Table 3: Machines used for comparison.

	Machine										
	#	Nature	Env.	CPU							
Solution				Name	Freq.	Gen.	Release	Arch.	Node	C / T	ISA
S^{3}OvA	S1-M1	Virtual[1]	UNIX	Intel(R) Xeon(R)	2.20 GHz	5	Q2 2016	Broadwell-EP	14 nm	2 / 4	x86-64 (32-bit, 64-bit)
	S1-M2	Physical	Windows	Intel core i3-2328M		2	Q1 2012	Sandy Bridge	32 nm		x86-64 (64-bit)
Edge2Train	S2-M1	Physical	UNIX	Intel core i7-5500U	2.40 GHz	5	circa 2015	Broadwell	14 nm	2 / 4	x86-64 (64-bit)
	S2-M2		Windows	Intel core i7-8650U	1.90 GHz (boosts up to 4.2 GHz)	8	circa 2017-2018	Kaby Lake-R		4 / 8	
`scikit-learn` [74]	S3-M1	Virtual[1]	UNIX	Intel(R) Xeon(R)	2.20 GHz	5	Q2 2016	Broadwell-EP	14 nm	2 / 4	x86-64 (32-bit, 64-bit)

$$H(NS, NF) = H_{init} - (NF - 1) \cdot 800 - \left(\frac{NS - 100}{50}\right) \cdot (800 + (NF - 1) \cdot 400) \quad (6)$$

where H_{init} represents the initial free memory prior to any allocation for the configuration (100, 1).

Although occasional variations have been observed, the model remains broadly reliable and enables the estimation of critical stability thresholds. Targeted cache memory tests, however, reveal a more constrained picture: critical allocation limits are reached as early as 200 $\times$ 200 samples, leading to systematic allocation failures and necessitating strict buffer sizing control alongside an algorithmically adapted strategy to maintain performance within available hardware bounds. Compounding this, the post-training memory footprint, whilst moderate, exhibits a linear growth of +160 B per increment of 50 samples, confirming the need for a dynamic and predictive memory management strategy to keep allocation under control beyond the learning phase. These constraints are further exacerbated by kernel parameter sensitivity, which heightens in a multiclass context due to the increased complexity of inter-class interactions, making careful tuning essential to avoid performance degradation. Taken together, these findings underscore the importance of meticulous embedded resource engineering and lead to a major conclusion: *whilst S^{3}OvA core demonstrates remarkable efficiency in binary classification, its extension to multiclass scenarios demands heightened attention to hardware ceilings and targeted algorithmic adaptations to preserve stability and portability guarantees under stringent constraints.*

Required adaptations The transition to multiclass classification is not merely a functional extension of the S^{3}OvA core, but rather a profound strategic reconfiguration. This shift rests on two closely interlinked pillars: targeted algorithmic optimisation and controlled memory resource management.

In this context, the precise tuning of parameters—particularly those related to kernel functions—proved to be a critical factor even during binary classification experiments. This criticality is significantly amplified in the multiclass setting, where the complexity of inter-class interactions gives rise to increased overlap and ambiguity. The careful selection of kernels, their fine-tuned parameterisation, and the associated regularisation criteria directly determine the S^{3}OvA core's ability to maintain effective class separation. This need for optimisation is substantiated by the in-depth analysis of errors observed in the binary case, which revealed non-negligible rates of false positives and false negatives. These errors point to structural shortcomings in the ability of certain models to establish clear decision boundaries. While such limitations may be partially tolerable in the binary context, they become highly problematic in multiclass scenarios, where decisions operate within a complex graph of interdependent class relationships.

Consequently, three strategic directions emerge as essential: (i) the application of advanced hyperparameter optimisation protocols to enable robust tuning in the face of data variability; (ii) the systematic exploration of highly discriminative kernels, better suited to managing multiple decision boundaries simultaneously; and (iii) the integration of specialised regularisation techniques capable of smoothing transitions between classes without compromising decision accuracy. Thus, algorithmic optimisation transcends mere performance enhancement and becomes a structural requirement, ensuring the stability, robustness, and scalability of the S^{3}OvA core.

In parallel with these algorithmic imperatives, the empirical analysis of memory behaviour in the binary case provides crucial technical benchmarks for guiding the transition to multiclass classification. Tests conducted with NF = 55 identified an acceptable equilibrium threshold below which memory usage remains manageable. Maximum measurements indicate that Flash memory usage reaches 25.40% of the space allocated for user code, while approximately 33% of the available dynamic memory is statically allocated to global variables at startup. While these figures suggest exploitable headroom, one parameter emerges as a major limiting factor: dynamic heap memory, which is heavily solicited as NS increases. The empirical equation (see Equation 6) highlights a rapid decrease in the initially available memory, with critical scenarios emerging. At NS = 700, a deficit of -29 128 B is observed, whereas at NS = 600, only 15 672 B remain available—insufficient to accommodate a simple 100×100 cache, which alone consumes 40 016 B. Conversely, at NS = 500, a stabilised threshold of 60 472 B of free memory emerges as an acceptable compromise, accounting for both context-dependent compiler optimisation effects and the post-training memory footprint.

These observations converge towards a structurally significant sizing decision: setting the operational limit at NS = 500. This equilibrium point maximises model scalability while strictly adhering to the hardware constraints of the platform. Additionally, the deactivation of the training cache is adopted as a complementary lever to free up memory, allowing available resources to be redirected towards multiclass classifiers and enhancing the overall efficiency of the system.

5.3.2 Data selection and experimental strategy

The construction of an appropriate experimental corpus constitutes a fundamental requirement for the rigorous evaluation of the multiclass extension of S^{3}OvA core. Our methodological approach is structured around the selection of balanced datasets, thereby ensuring robust and generalisable validation of our algorithm's performance. Experience gained from evaluating S^{3}OvA core in binary classification led us to adopt a balanced sizing strategy, working with sample sizes (NS $\in$ {100, 150, 200}), corresponding respectively to class distributions of (50/50), (75/75) and (100/100). Applying an 80/20 train/test split yielded training sets containing 40, 60, and 80 samples respectively.

This preliminary experience naturally directs us towards the multiclass extension, where the operational constraint set at NS = 500 (for NF = 55) imposes a crucial optimisation of the trade-off between the number of classes and the per-class training sample size. Comparative analysis reveals three possible configurations: allocating 100 samples per class accommodates only five classes, thus limiting the evaluation of multiclass scalability; 50 samples per class allow for ten classes but result in only 40 training samples per class, insufficient to capture the complexity of inter-class overlaps. Consequently, the configuration of 75 samples per class emerges as the optimal compromise, providing 60 training samples per class while enabling experimentation with six classifiers ($75 \times 6 = 450 < 500$) and even extension to seven classifiers (NS = 525), remaining close to the operational limit without significantly exceeding it.

Beyond this quantitative optimisation, experimental robustness requires a strategy of linear progression in the number of classes, enabling the scalability of S^{3}OvA core to be assessed under controlled conditions. However, this linear progression in the number of classes must be accompanied by a coherent evolution in feature dimensionality. Starting from the baseline configuration derived from the binary case, analysis yielded a mathematical relationship to guide extension to the multiclass scenario: for a number of classes (nc), the optimal dimensionality NF follows the rule:

$$NF = 2^{(nc-1)}.$$

This relationship captures the exponential growth in the complexity of the representation space required to maintain inter-class separability.

Applying this mathematical model yields a systematic progression, whereby the dimensionality increases from 4 features for 3 classes ($NF = 2^{(3-1)} = 4$) to 64 features for 7 classes ($NF = 2^{(7-1)} = 64$), passing through 8, 16, and 32 features for 4, 5, and 6 classes, respectively. This progression reflects the necessity of an exponentially richer representation space to capture inter-class distinctions in increasingly complex multiclass contexts.

On the basis of these theoretical considerations, we proceeded with the selection and adaptation of real-world datasets exhibiting varying levels of complexity for each experimental configuration. For the three-class configuration, we employed the Iris dataset (150,4), applying SMOTE to balance each class to 75 samples and produce a final corpus of (225,4). The progression to four classes led us to use the *Abalone* [77] dataset (4177,8), constructing a target variable *AgeClass* with four modalities {"*Young*", "*Young Adult*", "*Mature Adult*", "*Old*"} based on the age attribute, followed by rebalancing to obtain (300,8). For five classes, we selected the *Letter Recognition* [78] dataset (20000,16), extracting five specific letters, followed by selection and balancing to form a corpus of (375,16).

Complexity increases with the six-class configuration, where we adapted the *Dermatology* [79] dataset (366,34) via dimensionality reduction to 32 features, followed by SMOTE rebalancing of under-represented classes and sub-sampling of over-represented classes to obtain (450,32). For seven classes, we worked with the *Covertype* [80] dataset (581012,54), performing dimensional expansion through polynomial feature generation, followed by variance threshold-based feature selection to reach 64 dimensions, and finally constructing a balanced corpus (525,64). Finally, to explore a boundary case with eight classes, we used the MNIST dataset (70000,784), applying PCA-based dimensionality reduction to 128 components, then selecting eight of the ten available classes to produce a corpus of (600,128).

5.3.3 Deployment of S^{3}OvA in multiclass mode

The multiclass deployment of S^{3}OvA extends the S^{3}OvA core architecture to manage multiple binary classifiers while preserving the three pillars defined in Section 5.2.2—modular `C++` design, embedded compilation and execution, and real-time memory and performance optimisation. This extension maintains architectural consistency while addressing the added complexity of multiclass settings and anticipating the constraints of the Dynamic Update Case in operational environments.

Binary classifiers are constructed using the OvA strategy in two complementary modes: (i) batch training (see Figures 2(a) and 2(c)), enabling full pre-training on the complete dataset; and (ii) a simulated online mode based on Syed *et al.*'s method (see Figures 2(b) and 2(c)). The latter facilitates preparation for dynamic updates, controlled evaluation of memory impact, and performance comparison with batch training.

The implementation follows a five-component modular structure: `S3OvACore.h` manages binarised classifier training; `BatchOvAClassifier.h` encapsulates batch OvA learning; `IncrementalOvALearning.h` implements Syed *et al.*'s incremental method; `data.h` centralises execution data (features, labels, parameters); and `S3OvA.ino` serves as the embedded entry point. This organisation ensures readability, maintainability, and extensibility toward future learning strategies.

Multiclass scaling introduces substantial memory pressure due to the increased number of binary classifiers, requiring coordinated optimisation strategies. These include fine-grained memory monitoring with immediate deallocation of temporary structures; per-classifier metadata management (individual bias `_b`, sample counts, training states); isolation of internal data structures such as `y_internal` (binarised labels) and `_alphas` (dedicated Lagrange multiplier memory); and controlled allocation and prompt release of `y_binary` to minimise persistent memory usage.

5.3.4 Results and evaluation

The set of results generated through the deployment of S^{3}OvA in multiclass mode under the adopted progressive experimental strategy reveals the complex dynamics between dimensionality, number of classes, and the physical constraints of MCUs. This systematic progression, guided by the mathematical relation $NF = 2^{(nc-1)}$, has enabled an empirical evaluation of operational limits while characterising performance under controlled conditions of increasing complexity (see Figure 7; hardware details in Table 2 and Table 3).

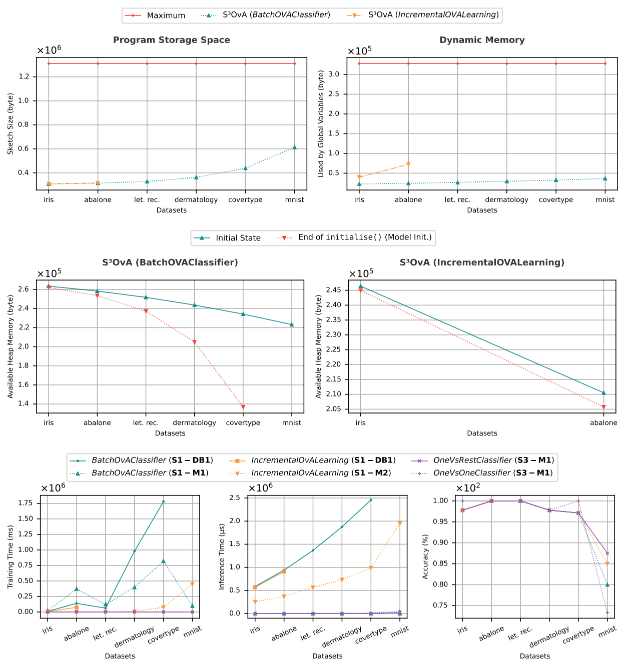


Figure 7: S^{3}OvA in multiclass mode: experimental results and assessment.

Within this rigorous experimental framework, the performance of S^{3}OvA in batch training mode, via the `BatchOVAClassifier` on the ESP32-S Module, demonstrates a remarkable capacity to maintain algorithmic accuracy despite severe hardware constraints. The exceptional accuracy of 97% to 100% achieved on the Iris (225×4), Abalone (300×8), Letter Recognition (375×16), Dermatology (450×32), and Covertype (525×64) datasets validates the robustness of the implementation against the exponential progression of dimensionality. This performance is accompanied by remarkably efficient memory management, with the PSS footprint evolving in a controlled manner from 307.14 kB (Iris) to 439.144 kB (Covertype), demonstrating that the algorithm effectively exploits implemented optimisations to maintain an acceptable footprint even in the most demanding operational configurations.

These accuracy performances are accompanied by a temporal analysis that highlights computational complexity consistent with the theory of multiclass SVMs: training times range from approximately 10 s on Iris to nearly 30 min on Covertype. This progression reflects the expected increase in computational load with dataset size and complexity. In parallel, inference times remain within near real-time ranges, from 587 µs to 2.5 ms, even under the most demanding configurations.

However, despite these promising performances, the operational boundary materialises critically with the MNIST dataset (600×128), provoking a *heap overflow* that precisely delineates the operating range of S^{3}OvA in batch training mode. This limitation reveals that the constraint does not reside in the number of classes per se, but rather in the complex interplay of dimensionality, sample size, and the memory architecture of the ESP32-S Module. Notably, the insights derived from the binary case—where an operational constraint was established at NS = 500 for NF = 55 (i.e., 500×55)—find empirical validation here. The successful extension to 7 classifiers with Covertype (525×64), remaining close to this limit without significantly exceeding it, confirms the robustness of the strategy initially developed for the binary case. This coherence demonstrates that the constraint NS $\times$ NF $\approx$ 27500 elements ($\approx$ 110 kB) indeed constitutes a fundamental hardware limit of the ESP32-S Module, irrespective of binary or multiclass algorithmic complexity. The failure on 8 classes with 128 dimensions ($600 \times 128 = 76800$), vastly exceeding this constraint, definitively validates that the relation $NF = 2^{(nc-1)}$ approaches the theoretical limit of the available hardware capabilities.

In parallel with this batch training analysis, the exploration of the simulated online learning mode through `IncrementalOvALearning` reveals a different and particularly instructive dynamic regarding memory constraints. Results on the ESP32-S Module show substantial temporal gains for Iris (5.4 vs 10 s) and Abalone (76 vs. 140 s), demonstrating the efficiency of the incremental approach. However, the significant increase in dynamic memory usage (from 24.324 kB to 72.34 kB for Abalone) precipitates the onset of memory overflows (*region* `dram0_0_seg` *overflowed*) from the Letter Recognition dataset (375×16). This crucial observation reveals that the limitation arises not from the number of classes (5 for Letter Recognition vs 4 for Abalone, which succeeds), but from the NS $\times$ NF combination that determines the memory footprint of incremental structures.

The progressive overflows observed—31.632 kB for Letter Recognition (375×16), 247.496 kB for Dermatology (450×32), 795.048 kB for Covertype (525×64), and 2.158 128 MB for MNIST (600×128)—perfectly illustrate this quadratic relationship between NS $\times$ NF and memory consumption. This quantitative characterisation establishes that incremental learning, while advantageous in terms of time, exhibits a memory consumption profile that rapidly exceeds the capacity of MCUs beyond modest configurations, regardless of the number of classes.

This quantitative characterisation of hardware limitations finds complementary validation in the evaluation of the incremental modality on physical machine (S1-M2) using Dev-C++[2], which confirms that these limitations stem exclusively from hardware constraints. The exceptional performances obtained—687 ms for Iris and sustained accuracy up to MNIST—demonstrate the algorithmic validity of the approach and its potential for deployment in less constrained environments. This cross-validation underlines that S^{3}OvA constitutes a generic solution, whose observed limitations faithfully reflect the physical boundaries of the target hardware.

To contextualise these results within the broader ML ecosystem, comparison with `scikit-learn` proves essential for establishing the relative positioning of S^{3}OvA. This comparative evaluation is based on a strict methodology ensuring fairness: the use of identical datasets, homogeneous parameters (`C = 1.0`, maximum 10^4 iterations, 80/20 partitioning), and identical κ per configuration.

A fundamental algorithmic difference nevertheless distinguishes the two solutions: `scikit-learn` leverages `LIBSVM` in the background, which implements an optimised SMO-type decomposition method proposed by Fan *et al.* for the `SVC` classifiers employed by `OneVsRestClassifier` and `OneVsOneClassifier`. In contrast, S^{3}OvA relies on an implementation of Platt's original SMO decomposition, used by `BatchOvAClassifier` and `IncrementalOvALearning`. This divergence in SMO implementation is accompanied by a critical environmental asymmetry: S^{3}OvA operates without memory caching in order to faithfully reflect embedded constraints, whereas `scikit-learn` exploits a 200 MB cache by default. This dual asymmetry—both algorithmic and environmental—far from biasing the analysis, highlights the magnitude of the challenge addressed by our implementation. The remarkable runtime performance of `scikit-learn` (14 to 180 ms for `OneVsRestClassifier`, 9 to 70 ms for `OneVsOneClassifier`) can be explained by Fan *et al.*'s advanced optimisation, substantial memory caching, and decades of refinement, yet remains unattainable in a resource-constrained context.

Nevertheless, the accuracy analysis reveals a remarkable consistency: S^{3}OvA achieves comparable, sometimes superior, performance, notably on MNIST where our solution achieves 80–85% against 73% for `OneVsOneClassifier`. This crucial observation demonstrates that adaptation to hardware constraints preserves, and in some cases enhances, the predictive quality of the algorithm.

5.4 Dynamic update case: validation of adaptive capabilities

5.4.1 Orchestration of adaptive capabilities

The strong performance of `BatchOVAClassifier` in static settings, together with the temporal validation of `IncrementalOVALearning` in less constrained

[2]Dev-C++ is a full-featured C/C++ IDE for Windows platforms: `https://www.bloodshed.net`.

environments, confirms the adaptive capabilities of S^{3}OvA. Beyond empirical validation, this complementarity extends the classical TinyML paradigm toward dynamic applications with continuously evolving models. `BatchOVAClassifier` ensures robust initial training under strict memory constraints, whereas `IncrementalOVALearning` enables progressive adaptation when resources allow, thereby validating the modular design and its suitability for complex, evolving scenarios.

Operationally, the central challenge lies in orchestrating these capabilities within a coherent framework. This requires explicit coordination and synchronisation mechanisms to enable controlled deployment and evaluation under real conditions, revealing the interactions between adaptive mechanisms and their environment. Conceptualising IoT architectures built upon S^{3}OvA thus becomes essential, ensuring effective adaptation to operational changes.

This orchestration is concretely realised through a structured workflow integrating the offline and online phases of S^{3}OvA (see Figures 2, 3, and 2(c)). Once deployed on an IoT platform, the system exhibits autonomous decision-making capable of distinguishing between: (i) familiar data requiring prediction only; (ii) new instances of known classes, potentially reflecting concept drift and requiring incremental adaptation; and (iii) data indicating the emergence of novel classes necessitating model extension.

The complete architecture of these future systems is illustrated by the workflow in Figure 8, which discloses the operational logic governing their adaptive behaviour from initialisation through to production. During the setup phase, the system first verifies the existence of a pre-trained model in storage: if found, it is loaded directly for immediate resumption of operations; otherwise, offline initial training is triggered, automatically selecting `S3OvACore` for binary classification or `BatchOVAClassifier/IncrementalOVALearning` for multiclass scenarios, both internally leveraging `S3OvACore` for fundamental operations.

Once operational, the system enters its main loop and continuously processes incoming data through the `Decision` module, which distinguishes three scenarios. When new data reside within the known statistical distribution and the neighbourhood of existing support vectors, the trained model is applied directly for prediction, optimising computational and temporal resources. When data conceptually belong to an existing class but exhibit drifted characteristics—typical of evolving IoT environments subject to temperature variations, sensor noise, or environmental changes—an incremental update is triggered, combining new examples with marginal support vectors to produce a more representative model saved for future predictions. Finally, upon detection of an unknown label, the most sophisticated process is activated: a new OvA classifier is created for the emerging class, endowing the system with structural evolvability that enables organic adaptation to its operational environment without external intervention.

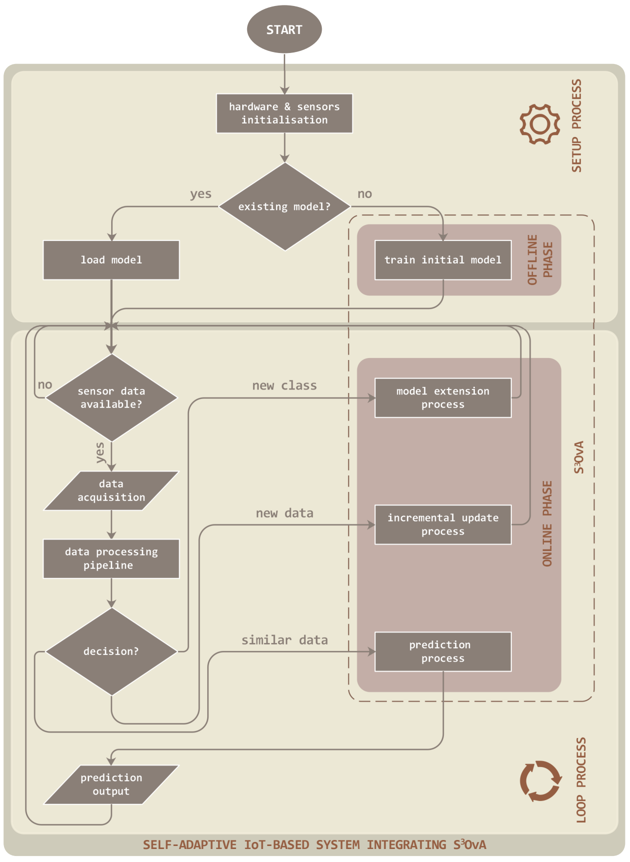


Figure 8: Architectural workflow of a self-adaptive IoT-based system integrating S^{3}OvA solution.

5.4.2 Data and evolutionary scenarios

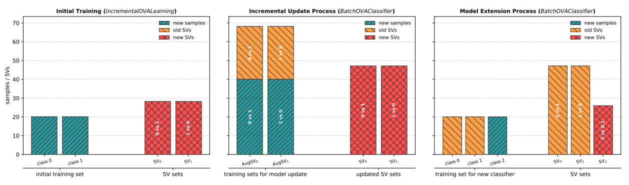
The evaluation of S^{3}OvA's adaptive capabilities requires a controlled experimentation framework reproducing the three principal evolutionary scenarios: (i) the reception of similar data, (ii) the onset of concept drift, and (iii) the emergence of new classes. To this end, the experimentation employs the *Auslan (Australian Sign Language) Signs (High Quality)* [81] dataset, a gesture-based corpus comprising nine subsets (`tctodd1`–`tctodd9`) and totalling 146949 vectors described by 22 dimensions. Each instance is enriched with the metadata *file*, *label*, and *sign*, thereby ensuring full traceability and enabling the use of *sign* as the target variable in dataset construction.

Within this framework, the experimental protocol begins with a binary reference configuration using a dataset of size (50×22), consisting of two balanced classes (0 and 1) partitioned according to an 80/20 ratio in order to establish the behavioural baseline of system. This initial configuration validates the standard *Prediction Process* on 10 test samples following the training of 40 samples.

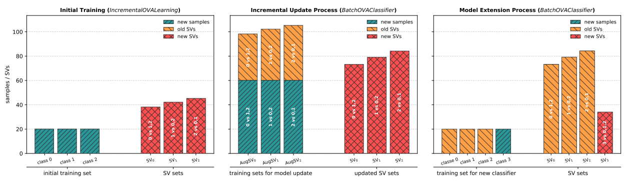
Once the baseline is established, concept drift is simulated by introducing a second dataset (50×22) that preserves the same classes but exhibits evolving feature distributions. The 40 new training samples, combined with the support vectors extracted during the initial training phase, drive the *Incremental Update Process*, thereby enabling a quantitative assessment of the performance improvement resulting from adaptation on the 10 remaining test samples.

Beyond this adaptation to internal variations, the emergence of new classes is simulated through the introduction of a dataset (25×22) corresponding to a previously unseen class (class 2). This extension triggers the construction of a new OvA classifier trained on 20 samples, balanced by an equal number of support vectors drawn from the existing classes. This configuration gradually transforms the initial binary model into a ternary architecture, empirically validating the structural evolutionary capacity of the approach.

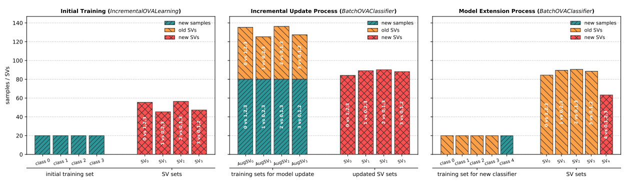
The protocol then proceeds through a controlled dimensional escalation: configuration (75×22) with 3 classes, followed by (100×22) with 4 classes, and continuing systematically until the next physical constraint is reached. At each stage, dataset balancing and the 80/20 partitioning are preserved, ensuring methodological consistency with the preceding evaluations.



(a) 1st experimental protocol.



(b) 2nd experimental protocol.



(c) 3rd experimental protocol.

Figure 9: Training and support vector sets across experimental protocols.

5.4.3 Deployment of S^{3}OvA in dynamic update mode

This phase completes the empirical validation of S^{3}OvA's adaptive capabilities (see Section 5.4.1), demonstrating real-time feasibility on the ESP32-S Module while preserving structural modularity (see Sections 5.2.2 and 5.3.3), incremental adaptation (see Figures 3(b) and 2(c)), and coordinated offline/online orchestration (see Figure 8).

In accordance with the previously defined workflow (see Figure 8), S^{3}OvA is deployed following the same two-phase operational logic. The ESP32-S mirrors this structure through its `setup()` and `loop()` cycles, ensuring strict continuity between the conceptual design and its embedded realisation.

Within this configuration, the execution flow directly activates `Decision.h`, which selects and executes the appropriate action within the tripartite architecture—either the *Prediction Process*, the *Incremental Update Process*, or the *Model Extension Process*.

5.4.4 Functional evaluation and performance

Building on the progressive experimental protocols established earlier in Section 5.4.2, this evaluation quantifies the adaptive capabilities of S^{3}OvA along three complementary axes: novelty detection, computational costs associated with dynamic updates, and the evolution of predictive capacities across the considered scenarios (see Figures 9 and 10).

At the outset, the first experimental protocol, based on an initial dataset of configuration 50×22 with two balanced classes, establishes the system's baseline metrics. The initial training on 40 samples with an RBF kernel ($\gamma = 100$) required 1.986 s, yielding a model achieving 90% accuracy on 10 test samples with an inference time of 75.349 ms. Subsequently, the incremental update, incorporating 40 new samples into the 56 pre-existing support vectors, required 6.436 s while maintaining accuracy at 90%, despite a slightly increased inference time (76.497 ms). Furthermore, extending the model to include a third class highlights the robustness of the OvA strategy. Training the new classifier, using a polynomial kernel of degree-3 with 20 dedicated samples for the new class versus the support vectors of the previous classes forming a balanced dataset, was completed in 3.920 s. The resulting accuracy of 86.67% on 20 test samples is particularly noteworthy given both the heterogeneity of kernels employed and the transition from a binary to a ternary architecture.

Moreover, evaluation on extended configurations demonstrates a substantial enhancement of adaptive capabilities. The second protocol (75×22, 3 classes) reveals a significant performance gain: incremental updating improved an initial accuracy of 80% to a perfect 100%, validating the effectiveness of the *Incremental Update Process*. This 20-point improvement is accompanied by increased computational cost (31.375 s), justified by the enlarged data volume (60 new samples, 125 support vectors). Similarly, the third protocol (100×22, 4 classes) confirms this trend with a 15-point improvement (80% to 95%) following incremental updating. In parallel, extension towards a fifth class sustains high accuracy at 88.46%, demonstrating the model's stability under progressive structural expansion.

Concurrently, memory resource analysis reveals a

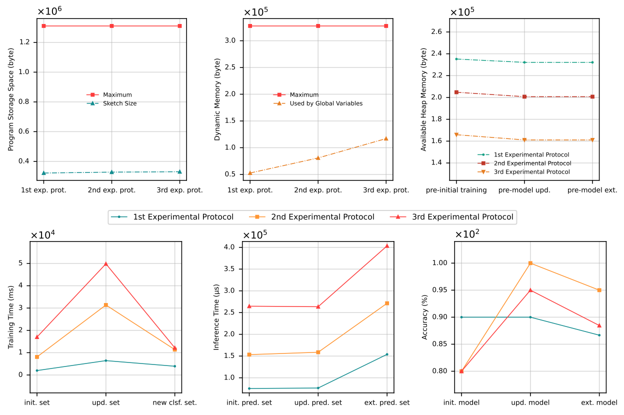


Figure 10: Performance metrics across experimental protocols.

particularly optimised implementation, despite cache being disabled. Across the three evaluated protocols, memory consumption exhibits remarkable stability. The available heap remained constant between model initialisation and its successive updates: 232 148 B (protocol 1), 200 724 B (protocol 2), and 161 112 B (protocol 3). This invariance confirms efficient memory management, preventing both leakage and fragmentation during adaptations. In parallel, PSS increased moderately from 321 796 B to 330 480 B across the protocols, while global variables evolved from 52 652 B to 117 044 B, reflecting the controlled increase in structural complexity.

Nevertheless, the fourth experimental protocol (125×22, 5 classes) exposes a new critical physical constraint of the system. The compilation error "`region `dram0_0_seg' overflowed by 36696 bytes`" indicates an overflow of the available memory capacity, thereby empirically establishing the operational limit of the system. This constraint complements the limitations previously identified in binary and multiclass configurations, thereby defining the complete operational envelope of S^{3}OvA on the ESP32-S Module.

6 Synthesis of results and improvement perspectives

S^{3}OvA establishes the viability of a reformable TinyML solution that coherently integrates SVMs, SMO decomposition, the Syed *et al.*'s method, and the OvA strategy directly on MCU. The experiments conducted on the ESP32-S Module highlight robust algorithmic performance, while also revealing essential margins for optimisation to enhance the efficiency of future self-adaptive IoT-based systems. Furthermore, the robustness of the solution is reinforced by effective validation on physical machines, thereby demonstrating its practical relevance beyond the experimental framework.

Indeed, the coherent integration of the components—SVM for classification, SMO for optimisation, the Syed *et al.*'s method for incremental adaptation, and OvA for multiclass extension—establishes a proven functional foundation. The binary-case performance empirically validates the robustness and portability of the algorithmic core, confirming the relevance of this methodological combination for embedded intelligence, while the implementation of polynomial and RBF kernels, in addition to the natural linear kernel, significantly strengthens the discriminative capacity.

Moreover, the efficiency of dynamic updating is manifested through substantial adaptive gains of up to 20 points, demonstrating its ability to evolve in response to environmental changes. This adaptability, combined with remarkable memory stability and only limited degradation during structural extensions (3.33 to 6.54 points), and the maintenance of very low inference latency suitable for near-instantaneous interactions—comparable to the best existing reformable TinyML solutions—attests to the architectural maturity required for deployment under real-world conditions.

Nevertheless, evaluation on the ESP32-S reveals critical limitations that guide future avenues for improvement. The memory overflow constraints ("`region `dram0_0_seg' overflowed`") observed from relatively modest configurations—$200 \times 55$ in binary case, $600 \times 128$ and $375 \times 16$ in the multiclass case with `BatchOvAClassifier` and `IncrementalOvALearning`, respectively; and 125×22 in in the dynamic update case—highlight the intrinsic limits of this particular platform. These observations do not call into question the algorithmic validity, but rather reveal the platform-specific constraints of the tested module.

Correspondingly, the required deactivation of caching in multiclass and dynamic update cases constitutes a critical indicator of the memory optimisations needed. This technical limitation suggests potential improvements in the management of data structures and the dynamic allocation of resources. The identification of the $\text{NS} \times \text{NF} \approx 27500$ elements (≈ 110 kB) relationship as a fundamental constraint provides a valuable engineering parameter for sizing future implementations.

Thus evidenced, S^{3}OvA positions itself as an algorithmically solid and functionally validated solution on physical machines, while also precisely identifying the hardware bottlenecks inherent to MCU execution. This balanced characterisation between demonstrated performance and identified limitations constitutes a robust methodological foundation for the evolution towards more efficient self-adaptive IoT-based systems, in which adaptive local intelligence becomes a driver of operational autonomy within constrained IoT ecosystems.

7 Discussion and future work

The experimental validation of S^{3}OvA across binary, multiclass, and dynamic settings confirms its feasibility while raising broader questions on adaptation under severe resource constraints. In Weyns' taxonomy (see Chapter 2 [9]), S^{3}OvA aligns primarily with *Learning from Experience*, yet its behaviour exhibits affinities with *Control-based Software Adaptation*, suggesting a hybrid paradigm combining incremental learning and control principles. The reported gains (up to +20 accuracy points at constant memory footprint) reflect stable adaptation under constraints, reminiscent of adaptive control frameworks providing formal guarantees, including Lyapunov-based methods [82], backstepping [83, 84], adaptive fuzzy control [85, 86], H_∞ approaches [87], and Discrete Controller Synthesis (DCS) [88]. However, unlike these continuous systems, S^{3}OvA lacks formal stability guarantees. Deriving stability certificates, Probably Approximately Correct (PAC) bounds under non-stationary distributions, and worst-case analyses under adversarial drift therefore constitutes a critical research priority for deployment in safety-critical contexts.

Performance is strongly conditioned by kernel selection: linear kernels favour stability, RBF kernels maximise accuracy at the expense of variance, and degree-3 polynomial kernels provide a balanced trade-off, echoing the aggressiveness–robustness dilemma in adaptive control. Automated kernel tuning and lightweight drift detection would enhance long-term adaptation, though memory-constrained pruning may induce catastrophic forgetting. Kernel-diverse ensemble variants of S^{3}OvA offer a promising mitigation strategy against gradual and abrupt distributional shifts.

S^{3}OvA does not natively process IoT streams; temporal alignment, fusion, and feature engineering are performed upstream, producing a unified feature vector on which SVM kernels capture inter-modal correlations. This architecture supports multi-source scenarios provided the constraint NS $\times$ NF $\approx$ 27500 is respected. Although more capable hardware platforms may push this boundary further, SMO complexity scales quadratically with NS, while increasing NF proportionally amplifies kernel computation costs. Beyond this regime, hybrid solutions combining model compression [20], intelligent subset selection [89], and sketching techniques [90] become necessary. Federated Learning and hierarchical edge–fog–cloud collaboration represent complementary directions, contingent upon integrated network–computation–learning co-design addressing communication, synchronisation, and security constraints.

Extending S^{3}OvA to embedded vision (compressed CNN descriptors), vibration-oriented acoustic analysis, or temporal prediction via Support Vector Regression (SVR) variants requires rigorous feature–classifier co-design and ecosystem-level standardisation (reference datasets, unified metrics, robust APIs (Application Programming Interfaces), certification frameworks for critical domains).

Overall, advancing theoretical guarantees, hyperparameter optimisation, sustained adaptation, distributed cooperation, domain expansion, and hardware–software co-design delineates a coherent roadmap toward a mature IoT-ready framework. The present results demonstrate that continuous adaptation under extreme resource efficiency is achievable, paving the way for reliable, autonomous intelligence at the network edge.

8 Conclusion

This paper demonstrates that an SVM architecture combining SMO, incremental learning based on Syed *et al.*'s method, and an OvA strategy operates effectively on MCUs, extending the technological frontier of TinyML. Unlike reformable TinyML solutions relying on offline full retraining—such as the SVM-based solution in [39], restricted to binary classification and requiring global parameter re-optimisation that overwrites prior knowledge—the proposed framework enables incremental multiclass adaptation. Its deployment on the ESP32-S module experimentally validates the migration of algorithms traditionally confined to high-performance computing environments toward ultra-constrained platforms.

The architecture's originality lies in sustaining strong discriminative performance under dynamic environmental conditions, thereby shifting TinyML from static inference toward continuous on-device learning. The coherent integration of heterogeneous kernels (linear, polynomial, RBF) further strengthens representational capacity while preserving computational feasibility.

Hardware constraints, rather than limiting factors, provide quantitative optimisation benchmarks. The reference memory footprint ($\approx$ 110 kB) and associated overflow thresholds define engineering invariants for dimensioning future adaptive systems, redirecting optimisation toward measurable objectives.

Beyond performance gains, this contribution questions prevailing IoT deployment models. Local learning eliminates cloud dependency, reducing latency, energy consumption, and security exposure, and repositioning edge devices as autonomous intelligent entities rather than passive sensors.

Accordingly, the proposed architecture—S^{3}OvA—establishes a trajectory toward intrinsically distributed artificial intelligence, where each IoT node develops adaptive capabilities in proximity to data, marking a significant conceptual advance in IoT system evolution.

References

[1] Ashton, K. (2009). That "Internet of Things" Thing. *RFID JOURNAL*. https://www.rfidjournal.com/expert-views/that-internet-of-things-thing/73881/.

[2] Sava, J.-A., Lueth, K. L., & Wegner, P. (2024). *State of IoT – Spring 2024* (p. 148) [Market Report]. IoT Analytics GmbH. https://iot-analytics.com/product/state-of-iot-spring-2024/.

[3] Aouedi, O., Vu, T.-H., Sacco, A., Nguyen, D. C., Piamrat, K., Marchetto, G., & Pham, Q.-V. (2024). A Survey on Intelligent Internet of Things: Applications, Security, Privacy, and Future Directions. IEEE Communications Surveys & Tutorials, 1–1. *IEEE Communications Surveys & Tutorials*. https://doi.org/10.1109/COMST.2024.3430368.

[4] Alkhabbas, F., Alsadi, M., Alawadi, S., Awaysheh, F. M., Kebande, V. R., & Moghaddam, M. T. (2022). ASSERT: A Blockchain-Based Architectural Approach for Engineering Secure Self-Adaptive IoT Systems. *Sensors*, *22*(18), 6842. https://doi.org/10.3390/s22186842.

[5] Sanchez-Iborra, R., & Skarmeta, A. F. (2020). TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. *IEEE Circuits and Systems Magazine*, *20*(3), 4–18. https://doi.org/10.1109/MCAS.2020.3005467.

[6] Abdelhaq, M. (2022). Internet of Things Fundamentals, Architectures, Challenges and Solutions: A Survey. International *Journal of Computer Science and Network Security*, *22*(1), 189–198. https://doi.org/10.22937/IJCSNS.2022.22.1.26.

[7] Tyagi, S., & Srivastava, A. (2024). Overcoming Terrain Challenges with Edge Computing Solutions: Optimizing WSN Deployments Over Obstacle Clad-Irregular Terrains. *Journal of Web Engineering*, 1127–1154. https://doi.org/10.13052/jwe1540-9589.2384.

[8] Ismail, S., Shah, K., Reza, H., Marsh, R., & Grant, E. (2021). Toward Management of Uncertainty in Self-Adaptive Software Systems: IoT Case Study. *Computers*, *10*(3), 27. https://doi.org/10.3390/computers10030027.

[9] Weyns, D. (2021). *An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective* (1st ed.). Wiley. https://doi.org/10.1002/9781119574910.

[10] Gheibi, O., Weyns, D., & Quin, F. (2020). Applying Machine Learning in Self-adaptive Systems: A Systematic Literature Review. *ACM Transactions on Autonomous and Adaptive Systems*, *15*(3), 1–37. https://doi.org/10.1145/3469440.

[11] Bureš, T. (2021). Self-Adaptation 2.0. *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 262–263. https://doi.org/10.1109/SEAMS51251.2021.00046.

[12] Casimiro, M., Romano, P., Garlan, D., Moreno, G. A., Kang, E., & Klein, M. (2021). Self-Adaptation for Machine Learning Based Systems. *ECSA-C 2021: Companion Proceedings of the 15th European Conference on Software Architecture*, *2978*. CEUR-WS.org.

[13] Banitalebi-Dehkordi, A., Vedula, N., Pei, J., Xia, F., Wang, L., & Zhang, Y. (2021). Auto-Split: A General Framework of Collaborative Edge-Cloud AI. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, KDD '21*, 2543–2553. https://doi.org/10.1145/3447548.3467078.

[14] Dastjerdi, A. V., & Buyya, R. (2016). Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer*, *49*(8), 112–116. https://doi.org/10.1109/MC.2016.245.

[15] Dutta, Dr. L., & Bharali, S. (2021). TinyML Meets IoT: A Comprehensive Survey. *Internet of Things*, *16*, 100461. https://doi.org/10.1016/j.iot.2021.100461.

[16] Merenda, M., Porcaro, C., & Iero, D. (2020). Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors*, *20*(9), 2533. https://doi.org/10.3390/s20092533.

[17] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198.

[18] Singh, R., & Gill, S. S. (2023). Edge AI: A Survey. *Internet of Things and Cyber-Physical Systems*, *3*, 71–92. https://doi.org/10.1016/j.iotcps.2023.02.004.

[19] Sipola, T., Alatalo, J., Kokkonen, T., & Rantonen, M. (2022). Artificial Intelligence in the IoT Era: A Review of Edge AI Hardware and Software. *2022 31st Conference of Open Innovations Association (FRUCT)*, 320–331. https://doi.org/10.23919/FRUCT54823.2022.9770931.

[20] Alqahtani, A., Xie, X., & Jones, M. W. (2021). Literature Review of Deep Network Compression. *Informatics*, *8*(4), 77. https://doi.org/10.3390/informatics8040077.

[21] Bhardwaj, K., Suda, N., & Marculescu, R. (2021). EdgeAl: A Vision for Deep Learning in the IoT Era. *IEEE Design & Test*, *38*(4), 37–43. https://doi.org/10.1109/MDAT.2019.2952350.

[22] Warden, P., & Situnayake, D. (2019). *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers* (1st ed.). O'Reilly Media, Inc.

[23] Maoui, M., & Benaboud, R. (2024). Perspectives of TinyML-Based Self-management in IoT-Based Systems. In M. R. Laouar, V. E. Balas, V. Piuri, D. Rad, Z. Touati Hamad, & A. Cheddad (Eds.), *13th International Conference on Information Systems and Advanced Technologies "ICISAT 2023": New Trends in Artificial Intelligence, Computing and Decision Making.* (Vol. 2, pp. 1–9). Springer. https://doi.org/10.1007/978-3-031-60594-9_1.

[24] Ravaglia, L., Rusci, M., Nadalini, D., Capotondi, A., Conti, F., & Benini, L. (2021). A TinyML Platform for On-Device Continual Learning with Quantized Latent Replays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *11*(4), 789–802. https://doi.org/10.1109/JETCAS.2021.3121554.

[25] Rajapakse, V., Karunanayake, I., & Ahmed, N. (2023). Intelligence at the Extreme Edge: A Survey on Reformable TinyML. *ACM Computing Surveys*, 3583683. https://doi.org/10.1145/3583683.

[26] Capogrosso, L., Cunico, F., Cheng, D. S., Fummi, F., & Cristani, M. (2024). A Machine Learning-Oriented Survey on Tiny Machine Learning. *IEEE Access*, *12*, 23406–23426. https://doi.org/10.1109/ACCESS.2024.3365349.

[27] Gama, J. M., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A Survey on Concept Drift Adaptation. *ACM Computing Surveys*, *46*(4). https://doi.org/10.1145/2523813.

[28] Platt, J. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. *Microsoft*, *MSR-TR-98-14*. https://www.microsoft.com/en-us/research.

[29] Galar, M., Fernández, A., Barrenechea, E., Bustince, H., & Herrera, F. (2011). An Overview of Ensemble Methods for Binary Classifiers in Multi-Class Problems: Experimental Study on One-vs-One and One-vs-All Schemes. *Pattern Recognition*, *44*(8), 1761–1776. https://doi.org/10.1016/j.patcog.2011.01.017.

[30] Syed, N. A., Liu, H., & Sung, K. K. (1999, August 31). Incremental Learning with Support Vector Machines. *Proceedings of the Workshop on Support Vector Machines at the International Joint Conference on Articial Intelligence*. IJCAI'99: The Sixteenth International Joint Conference on Artificial Intelligence, City Conference Center, Stockholm, Sweden.

[31] Syed, N. A., Liu, H., & Sung, K. K. (1999). Handling Concept Drifts in Incremental Learning With Support Vector Machines. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 317–321. https://doi.org/10.1145/312129.312267.

[32] Lawal, I. A. (2019). Incremental SVM Learning: Review. In M. Sayed-Mouchaweh (Ed.), *Learning from Data Streams in Evolving Environments: Methods and Applications* (Vol. 41, pp. 279–296). Springer. https://doi.org/10.1007/978-3-319-89803-2_12.

[33] *NodeMCU-32 Specification (V1.3)*. (2020). Shenzhen Ai-Thinker Technology Co., Ltd. https://docs.ai-thinker.com/_media/nodemcu32-s_specification_v1.3.pdf.

[34] *ESP32-S Specification (V1)*. (2019). Shenzhen Ai-Thinker Technology Co., Ltd. https://docs.ai-thinker.com/_media/esp32-s_specification.pdf.

[35] *ESP32 Series: Datasheet (Version 4.6)*. (2024). Espressif Systems (Shanghai) Co., Ltd. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[36] Benatti, S., Montagna, F., Kartsch, V., Rahimi, A., Rossi, D., & Benini, L. (2019). Online Learning and Classification of EMG-Based Gestures on a Parallel Ultra-Low Power Platform Using Hyperdimensional Computing. *IEEE Transactions on Biomedical Circuits and Systems*, *13*(3), 516–528. https://doi.org/10.1109/TBCAS.2019.2914476.

[37] Cai, H., Gan, C., Zhu, L., & Han, S. (2020). TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. In *Advances in Neural Information Processing Systems 33: 34th Conference on Neural Information Processing Systems (NeurIPS 2020)* (Vol. 33, pp. 11285–11297). Neural Information Processing Systems Foundation, Inc. (NeurIPS). https://proceedings.neurips.cc/paper_files/paper/2020.

[38] Pellegrini, L., Graffieti, G., Lomonaco, V., & Maltoni, D. (2020). Latent Replay for Real-Time Continual Learning. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 10203–10209. https://doi.org/10.1109/IROS45743.2020.9341460.

[39] Sudharsan, B., Breslin, J. G., & Ali, M. I. (2020). Edge2Train: A Framework to Train Machine

Learning Models (SVMs) on Resource-Constrained IoT Edge Devices. *Proceedings of the 10th International Conference on the Internet of Things*, 1–8. https://doi.org/10.1145/3410992.3411014.

[40] Disabato, S., & Roveri, M. (2020). Incremental On-Device Tiny Machine Learning. *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, 7–13. https://doi.org/10.1145/3417313.3429378.

[41] Lee, S., & Nirjon, S. (2020). Learning in the Wild: When, How, and What to Learn for On-Device Dataset Adaptation. *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, 34–40. https://doi.org/10.1145/3417313.3429382.

[42] de Prado, M., Rusci, M., Donze, R., Capotondi, A., Monnerat, S., and, L. B., & Pazos, N. (2021). Robustifying the Deployment of TinyML Models for Autonomous Mini-Vehicles. *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401154.

[43] Ren, H., Anicic, D., & Runkler, T. A. (2021). TinyOL: TinyML with Online-Learning on Microcontrollers. *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN52387.2021.9533927.

[44] Avi, A., Albanese, A., & Brunelli, D. (2022). Incremental Online Learning Algorithms Comparison for Gesture and Visual Smart Sensors. *2022 International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN55064.2022.9892356.

[45] Ren, H., Li, X., Anicic, D., & Runkler, T. A. (2024). On-device Online Learning and Semantic Management of TinyML Systems. *ACM Transactions on Embedded Computing Systems*, *23*(4), 1–32. https://doi.org/10.1145/3665278.

[46] Sudharsan, B., Yadav, P., Breslin, J. G., & Intizar Ali, M. (2021). Train++: An Incremental ML Model Training Algorithm to Create Self-Learning IoT Devices. *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, 97–106. https://doi.org/10.1109/SWC50871.2021.00023.

[47] Sudharsan, B., Breslin, J. G., & Intizar Ali, M. (2021). Imbal-OL: Online Machine Learning from Imbalanced Data Streams in Real-world IoT. *2021 IEEE International Conference on Big Data (Big Data)*, 4974–4978. https://doi.org/10.1109/BigData52589.2021.9671765.

[48] Sudharsan, B., Breslin, J. G., & Ali, M. I. (2022). ML-MCU: A Framework to Train ML Classifiers on MCU-Based IoT Edge Devices. *IEEE Internet of Things Journal*, *9*(16), 15007–15017. https://doi.org/10.1109/JIOT.2021.3098166.

[49] Pavan, M., Ostrovan, E., Caltabiano, A., & Roveri, M. (2023). TyBox: An Automatic Design and Code-Generation Toolbox for TinyML Incremental On-Device Learning. *ACM Transactions on Embedded Computing Systems*, 3604566. https://doi.org/10.1145/3604566.

[50] Rüb, M., Tuchel, P., Sikora, A., & Mueller-Gritschneder, D. (2024). A Continual and Incremental Learning Approach for TinyML On-device Training Using Dataset Distillation and Model Size Adaption. *2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS)*, 1–8. https://doi.org/10.1109/ICPS59941.2024.10639989.

[51] Anthony, G., Gregg, H., & Tshilidzi, M. (2007). Image Classification Using SVMs: One-against-One Vs One-against-All. *Proceedings of the 28th Asian Conference on Remote Sensing 2007 (ACRS 2007)*. 28th Asian Conference on Remote Sensing 2007 (ACRS 2007), Kuala Lumpur, Malaysia.

[52] Ruping, S. (2001). Incremental Learning With Support Vector Machines. *Proceedings 2001 IEEE International Conference on Data Mining*, 641–642. https://doi.org/10.1109/ICDM.2001.989589.

[53] Klinkenberg, R., & Joachims, T. (2000). Detecting Concept Drift with Support Vector Machines. *Proceedings of the Seventeenth International Conference on Machine Learning*, 487–494.

[54] Vapnik, V. N. (2006). *Estimation of Dependences Based on Empirical Data* (S. Kotz, Trans.; [2nd enl. ed.]. Reprint of 1982 ed. with afterword of 2006). Springer.

[55] Platt, J. C. (1999). Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In *Advances in Kernel Methods: Support Vector Learning* (pp. 185–208). MIT Press.

[56] Parnas, D. L. (1972). On the Criteria to Be Used in Decomposing Systems Into Modules. *Communications of the ACM*, 15(12), 1053–1058. https://doi.org/10.1145/361598.361623.

[57] Lilienthal, C. (2022). Improve Your Architecture with the Modularity Maturity Index. In *Software Architecture Metrics: Case Studies to Improve the Quality of Your Architecture* (First Edition, pp. 49–64). O'Reilly Media, Inc.

[58] Ford, N., Richards, M., Sadalage, P., & Dehghani, Z. (2021). *Software Architecture: The Hard Parts* (First Edition). O'Reilly Media, Inc.

[59] Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach* (First Edition). O'Reilly Media, Inc.

[60] Sarkar, S., Kak, A. C., & Rama, G. M. (2008). Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software. *IEEE Transactions on Software Engineering*, *34*(5), 700–720. https://doi.org/10.1109/TSE.2008.43.

[61] Coulin, T., Detante, M., Mouchère, W., & Petrillo, F. (2019). *Software Architecture Metrics: A Literature Review* (No. arXiv:1901.09050). arXiv. https://doi.org/10.48550/arXiv.1901.09050.

[62] Stroustrup, B. (Director). (2012). Style C++11 [Broadcast]. In *GoingNative 2012*. https://learn.microsoft.com/en-us/shows/goingnative-2012/keynote-bjarne-stroustrup-cpp11-style.

[63] Espressif Systems. (2024). *ESP32: ESP-IDF Programming Guide* (Version 5.3). Espressif Systems. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/esp-idf-en-v5.3-esp32.pdf.

[64] *make_classification*. (n.d.). Scikit-Learn. Retrieved June 30, 2025, from https://scikit-learn/stable/modules/generated/sklearn.datasets.make_classification.html.

[65] Haberman, S. (1976). *Haberman's Survival* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5XK51.

[66] Lohweg, V. (2012). *Banknote Authentication* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C55P57.

[67] Becker, B., & Kohavi, R. (1996). *Adult* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5XW20.

[68] Wolberg, W., Mangasarian, O., Street, N., & Street, W. (1993). *Breast Cancer Wisconsin (Diagnostic)* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B.

[69] Hopkins, M., Reeber, E., Forman, G., & Suermondt, J. (1999). *Spambase* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C53G6X.

[70] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. https://doi.org/10.1613/jair.953.

[71] Fisher, R. A. (1936). *Iris* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76.

[72] LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). *The MNIST Database of Handwritten Digits* [Dataset]. http://yann.lecun.com/exdb/mnist/.

[73] Fan, R.-E., Chen, P.-H., & Lin, C.-J. (2005). Working Set Selection Using Second Order Information for Training Support Vector Machines. *J. Mach. Learn. Res.*, *6*, 1889–1918.

[74] *scikit-learn: Machine learning in Python—Scikit-learn 1.7.1 documentation*. (n.d.). Retrieved August 26, 2025, from https://scikit-learn.org/stable/.

[75] Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, *2*(3), 1–27. https://doi.org/10.1145/1961189.1961199.

[76] *SVC*. (n.d.). Scikit-Learn. Retrieved July 4, 2025, from https://scikit-learn/stable/modules/generated/sklearn.svm.SVC.html.

[77] Nash, W., Sellers, T., Talbot, S., Cawthorn, A., & Ford, W. (1994). *Abalone* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C55C7W.

[78] Slate, D. (1991). *Letter Recognition* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5ZP40.

[79] Ilter, N., & Guvenir, H. (1998). *Dermatology* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5FK5P.

[80] Blackard, J. (1998). *Covertype* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C50K5N.

[81] Kadous, M. (2002). *Australian Sign Language Signs (High Quality)* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5VG6R.

[82] Zouari, F., Ben Saad, K., & Benrejeb, M. (2012). Robust Neural Adaptive Control for a Class of Uncertain Nonlinear Complex Dynamical Multivariable Systems. *International Review on Modelling and Simulations*, 5(5), 2075–2103.

[83] Zouari, F., Ben Saad, K., & Benrejeb, M. (2013a). Adaptive Backstepping Control for a class of Uncertain Single Input Single Output Nonlinear Systems. *10th International Multi-Conferences on Systems, Signals & Devices 2013 (SSD13)*, 1–6. https://doi.org/10.1109/SSD.2013.6564134.

[84] Zouari, F., Ben Saad, K., & Benrejeb, M. (2013b). Adaptive Backstepping Control for a Single-Link Flexible Robot Manipulator Driven by a DC Motor. *2013 International Conference on Control, Decision and Information Technologies (CoDIT)*, 864–871. https://doi.org/10.1109/CoDIT.2013.668965.

[85] Boulkroune, A., Hamel, S., Zouari, F., Boukabou, A., & Ibeas, A. (2017). Output-Feedback Controller Based Projective Lag-Synchronization of Uncertain Chaotic Systems in the Presence of Input Nonlinearities. *Mathematical Problems in Engineering*, 2017(1), 8045803. https://doi.org/10.1155/2017/8045803.

[86] Boulkroune, A., Zouari, F., & Boubellouta, A. (2025). Adaptive Fuzzy Control for Practical Fixed-Time Synchronization of Fractional-Order Chaotic Systems. *Journal of Vibration and Control*, 10775463251320258. https://doi.org/10.1177/10775463251320258.

[87] Rigatos, G., Abbaszadeh, M., Sari, B., Siano, P., Cuccurullo, G., & Zouari, F. (2023). Nonlinear Optimal Control for a Gas Compressor Driven by an Induction Motor. *Results in Control and Optimization*, *11*, 100226. https://doi.org/10.1016/j.rico.2023.100226.

[88] Gatouillat, A., Badr, Y., & Massot, B. (2018). QoS-Driven Self-adaptation for Critical IoT-Based Systems. In L. Braubach, J. M. Murillo, N. Kaviani, M. Lama, L. Burgueño, N. Moha, & M. Oriol (Eds.), *Service-Oriented Computing – ICSOC 2017 Workshops* (Vol. 10797, pp. 93–105). Springer International Publishing. ttps://doi.org/10.1007/978-3-319-91764-1_8.

[89] Mourad, S., Tewfik, A., & Vikalo, H. (2017). Data Subset Selection for Efficient SVM Training. *2017 25th European Signal Processing Conference (EUSIPCO)*, 833–837. https://doi.org/10.23919/EUSIPCO.2017.8081324.

[90] Gribonval, R., Chatalic, A., Keriven, N., Schellekens, V., Jacques, L., & Schniter, P. (2021). Sketching Data Sets for Large-Scale Learning: Keeping Only What You Need. *IEEE Signal Processing Magazine*, *38*(5), 12–36. https://doi.org/10.1109/MSP.2021.3092574.