JOURNAL OF SOFT COMPUTING AND DATA MINING
e-ISSN: 2716-621X

JSCDM

Vol. 6 No. 3 (2025) 85-102
https://publisher.uthm.edu.my/ojs/index.php/jscdm

On-Device Training Based Anomaly Detection Platform

Bao-Toan Thai[1], Vy-Khang Tran[1], Chi-Ngon Nguyen[1,2], Van-Khanh Nguyen[1,2*]

[1] *PLC Technology and IIoT Laboratory, College of Engineering, Can Tho University, Can Tho city, 94000, VIETNAM*
[2] *College of Engineering, Can Tho University, Can Tho city 94000, VIETNAM*

*Corresponding Author: vankhanh@ctu.edu.vn
DOI: https://doi.org/10.30880/jscdm.2025.06.03.007

Article Info

Received: 4 May 2025
Accepted: 13 August 2025
Available online: 30 December 2025

Keywords

On-device training, autoencoder, anomaly detection, microcontroller unit, tiny machine learning (tinyML).

Abstract

Anomaly detection (AD) identifies issues in monitored systems, allowing timely responses. It is used in areas like industrial systems, healthcare, and networks. Among various AD methods, Autoencoder (AE)-based models are popular for their unsupervised training, which doesn’t require labeled data. This study proposes a full platform that implements AE-based AD on tiny devices and tests it on motor operating noise. The hardware includes two microcontroller units (MCUs) responsible for (1) real-time data acquisition and processing, (2) AE training and threshold setting, and (3) running inference for anomaly alerts. To boost training speed, the platform supports both stochastic gradient descent (SGD) and adaptive moment estimation (Adam). The algorithm uses real-time data instead of pre-collected data to improve practical performance. Tests show the platform effectively trains normal motor noise and accurately sets the anomaly threshold. It achieves a real-time anomaly detection accuracy of 100%. The system runs fully automatically, making it suitable for integration into edge artificial intelligence of things (AIoT) systems.

1. Introduction

Anomaly recognition identifies atypical patterns in data. Anomaly detection (AD) methods detect patterns that significantly differ from normal data [1], playing a vital role in fields like healthcare [2], cybersecurity [3], and industry [4] by enabling timely interventions and improving system reliability.

Autoencoder (AE) models are widely used for AD in domains such as industry [5, 6], medicine [7, 8], and security [9, 10]. Most studies use large, resource-intensive, and costly devices. Some research addresses AD on resource-constrained devices using tiny machine learning (TinyML) platforms [11-13], where models are trained and fine-tuned on computers, then quantized to fit on small devices. The main drawback of this approach is that deployed models lack retraining capabilities, making it difficult to adapt to changes in the data. Since data distribution can evolve due to user behavior, environmental conditions, or other factors [14], static models may lose accuracy over time. Therefore, on-device training (ODT) for autoencoders is essential to overcome this limitation in TinyML applications.

Several studies have developed anomaly detection using on-device training but mainly deploy it on field-programmable gate arrays (FPGAs). Integrating machine learning into FPGA hardware offers advantages such as rapid computation and flexible upgrades [15-17]. Methods like extreme learning machines (ELM) and online sequential ELM (OS-ELM) on FPGAs have shown high-speed processing and resource efficiency, especially in anomaly detection, classification, and medical diagnostics [18-27]. These models exhibit low latency, making them suitable for real-time applications. However, most studies focus on evaluating performance without proposing a complete system. They often lack modules for data collection, processing, and a statistical mechanism for setting detection thresholds. A key limitation is the complexity of FPGA design, which requires expertise and technical resources. Additionally, the high cost of FPGAs limits their use in large-scale deployments like the Internet of Thing (IoT). Thus, implementing and evaluating on-device training for anomaly detection on low-cost microcontroller units (MCUs) is both important and practical.

This research proposes an ODT-based platform for AD that operates entirely on an embedded system, seamlessly integrating data collection, processing, transformation, training, statistical thresholding, and real-time model execution directly on the embedded system itself. An artificial intelligence (AI) library developed by the research team [28], named NeuronLite, is upgraded to support real-time training of an autoencoder-based anomaly detection model and has been successfully implemented across various MCU platforms. After that, the proposed platform has been applied to detect anomalies in the operation noise of a three-phase alternating current (AC) motor. The results not only highlight the system's strong performance in detecting real-world anomalies but also demonstrate its efficiency in resource usage, cost-effectiveness, and ease of deployment. The library further enables on-device model retraining when necessary, offering flexible and efficient performance optimization.

2. Methodology

To address the AD problem, the input data is first fed into an AE model for training, followed by thresholding to identify anomalies based on the learned representations, as illustrated in Fig. 1. Traditionally, implementing this process on a MCU relies on TinyML: the model is trained on a personal computer (PC), converted into a lightweight version, and then deployed to the MCU. However, whenever new data becomes available, the entire model must be retrained, which is time-consuming and complicates real-world deployment. To overcome these limitations, this study proposes a method that integrates the entire AD pipeline directly on the MCU, including training, inference, and automatic threshold selection. Finally, the proposed system is applied to detect anomalies in the noise generated by a three-phase motor.

2.1 The Platform for Creating and Training AI Models

2.1.1 Library NeuronLite

The study in [28] developed an artificial neural network (ANN) library for linear regression using the stochastic gradient descent (SGD) optimizer on MCUs. However, storing weights directly in the ESP-32's flash memory limited scalability when multiple models were needed for testing and comparison. To address this, we redesigned the library as NeuronLite, with updated classes, integrated the Adam optimizer, and added methods for model saving and loading. The goal is to provide a flexible, general-purpose library for tasks like regression, classification, and anomaly detection. It enables rapid deployment with minimal code, enhancing usability.

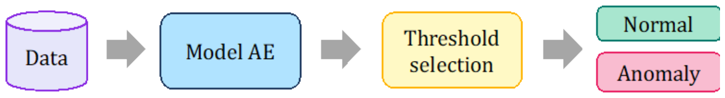


Fig. 1 *AD problem on computer*

Fig. 2 shows the structure of the NeuronLite library, which includes three main classes, each with a specific role in training and prediction. The NeuralNetwork class manages the overall model and contains multiple layers, either FCLayer or ActivationLayer. During execution, input data passes through each layer, where it is transformed into predicted outputs. These outputs are compared with actual values to compute errors, which are then used to update weights and biases via backward propagation. The FCLayer class computes outputs from inputs using weights and biases, which are updated during training by optimizers like Adam or SGD. The ActivationLayer applies nonlinear activation functions to the FCLayer output, enabling the network to learn complex patterns. It supports both forward and backward operations to ensure proper weight adjustment. Mathematical expressions for forward propagation, backpropagation, the SGD optimizer, and the mean squared error (MSE) loss function are detailed in [28]. The NeuronLite library has been extended with the Adam optimizer, which combines the strengths of root mean square propagation (RMSProp) [29] and SGD with momentum (SGDM) [30]. This improves optimization efficiency and convergence, offering clear advantages for deep learning models [31].

Adam utilizes two intermediate variables to estimate the momentum and the uncentered variance (second moment) of gradients at each update step. Specifically, Adam maintains two variables—first-order and second-order moments—to track the historical gradients of each weight. This design allows for more adaptive and efficient updates, contributing to the optimizer's robustness and effectiveness.

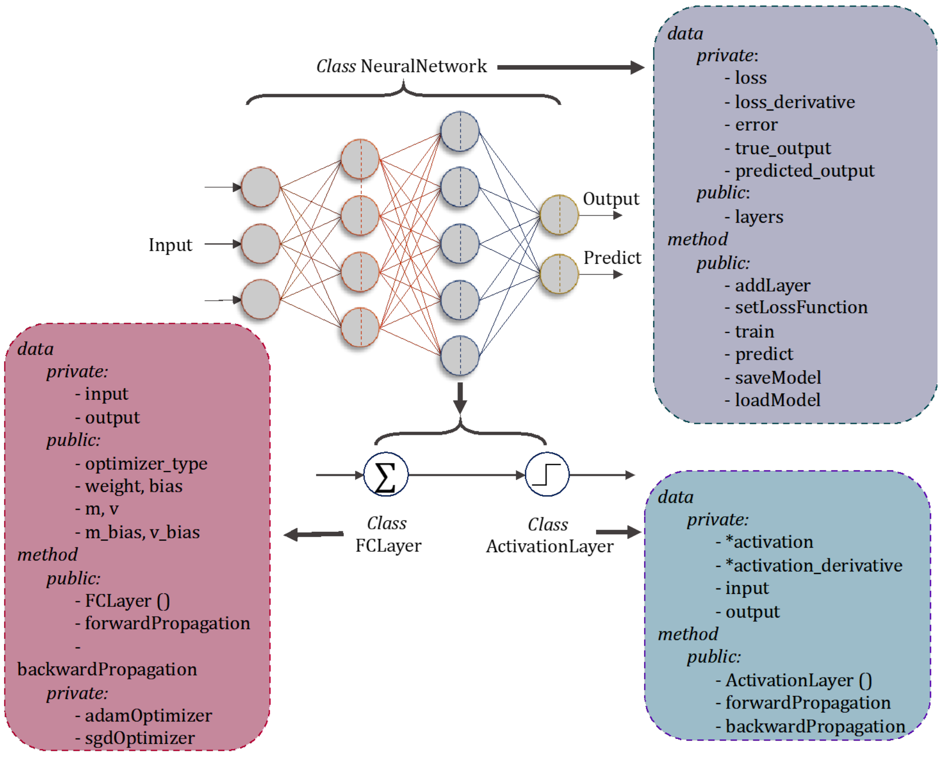


Fig. 2 *Overview of the NeuronLite library*

The first moment, denoted as m_t, represents the momentum of the gradient and is calculated by combining the previous moment value m_{t-1} with the current gradient g_t, using the formula:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

Where β_1 is the momentum parameter (typically set to 0.9), g_t is defined as follows:

$$g_t = \frac{\partial L}{\partial w_t} \tag{2}$$

Here, g_t is the gradient of the loss function with respect to the weights at time t. The gradient term g_t represents the direction and rate of change of the loss function L with respect to the model's parameters. It indicates how much and in which direction the parameters should be adjusted in order to minimize the loss.

The second moment, denoted as v_t, represents the uncentered variance (second moment) of the gradient and is calculated similarly, but using the square of the gradient:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3}$$

Where β_2 is typically set to 0.999, controlling the update rate of the variance variable and allowing the algorithm to adapt to changes in the gradient over time. Since both moments may be biased due to initialization at zero, they need to be bias corrected. The corrected first moment is calculated using the formula:

$$\widehat{m} = \frac{m_t}{1 - \beta_1^t} \tag{4}$$

And the corrected second moment is calculated using the formula:

$$\hat{v} = \frac{v_t}{1 - \beta_2^t} \tag{5}$$

After obtaining $\widehat{m_t}$ and $\widehat{v_t}$, Adam updates the model's weights using the following formula:

$$\theta_{t+1} = \theta_t - \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon} \tag{6}$$

Where α is the learning rate, typically adjusted to control the model's learning speed, and ϵ is a very small value (usually 10^{-8}) to prevent division by zero errors.

2.1.2 Setup Autoencoder Model

An AE is a neural network designed to encode and decode data. It learns to compress input into a simpler form and reconstruct it [32]. The goal is to reproduce the output to closely match the input. An AE has two main parts: an encoder and a decoder, connected by a latent space layer [13]. The encoder compresses the input; the decoder reconstructs it. A key feature is its symmetrical structure: input and output layers have the same number of neurons, while the latent layer has fewer neurons for effective compression [33].

Fig. 3 illustrates the AE structure used for anomaly detection, featuring an input layer of 576 neurons and a 2-neuron latent layer that compresses data into its simplest representation. The decoder reconstructs the input, forming a symmetrical [576–2–576] architecture. All layers employ the ReLU activation function to capture nonlinear relationships, and the model is trained by minimizing the MSE loss to reduce reconstruction errors. Given the constrained computational resources of platforms such as the ESP32, a complexity analysis was performed to assess feasibility. Each fully connected layer has a computational cost of $O(n_{in}\times n_{out})$, resulting in a total time complexity of approximately $O(2{,}304)$ per layer. The model includes 2,882 trainable parameters, requiring approximately 11 KB of memory in 32-bit floating-point format, with an overall inference footprint of around 83.5 KB as measured on the ESP32. These characteristics confirm that the proposed AE is lightweight, computationally efficient, and suitable for real-time deployment on ESP32-based systems.

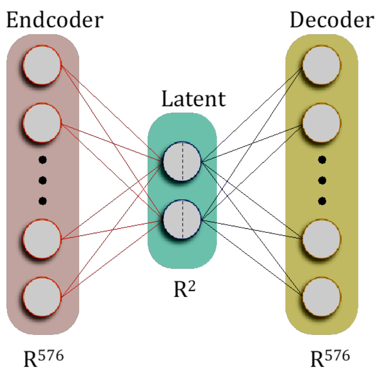


Fig. 3 *Model AE structure*

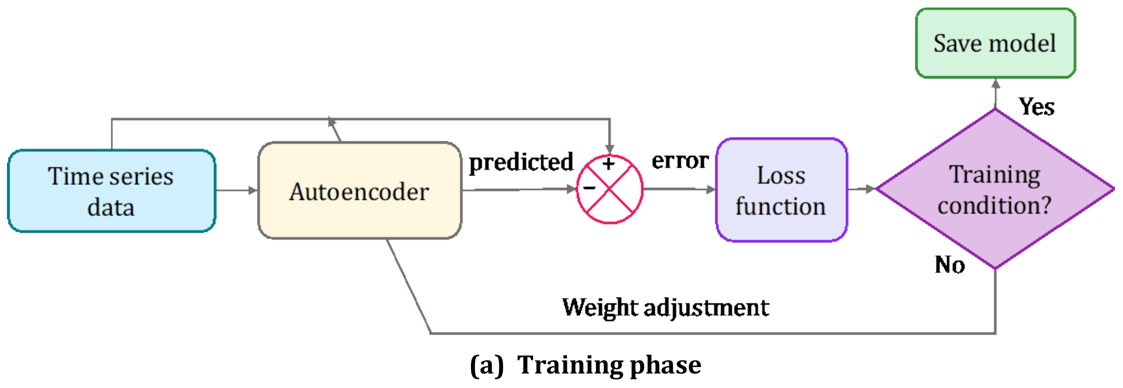
2.1.3 Training and Automatic Thresholding

The training process is illustrated in Fig. 4 (a). First, the time-series data will be directly fed into the AI network. At this stage, the AI learns the time-series data, identifying anomalies based on the error between the original and reconstructed data using the loss function. Finally, the AE model's training process is evaluated based on two criteria. The first criterion is training time: the model is trained for a predetermined time limit. If the training time has not been reached, the process continues; otherwise, once the time limit is met, the model automatically saves the current results. The second criterion is the loss value: if the error between the original and reconstructed data drops below a specified threshold, the training process terminates, and the satisfactory model is saved. Otherwise, the training continues until the desired performance is achieved. In practice, the selection between the two criteria depends largely on the characteristics of the dataset. For datasets that exhibit high stability and minimal temporal variation, the training time criterion is generally more appropriate. In such cases, the model can be trained within a limited time frame and still capture the essential features of the data. Conversely, for datasets with significant variability or complex nonlinear patterns, extended training periods—ranging from several days to even months—may be required for the model to adequately learn the complex structure. Therefore, the loss value criterion is considered more suitable for these types of data, as it allows training to continue until a satisfactory level of reconstruction accuracy is achieved.

After completing the training process, the AE model transitions to the predicting phase, as illustrated in Fig. 4 (b). This process is similar to the training phase: time-series data is fed directly into the trained AE model, which begins making predictions and calculates the error between the original data and the predicted data using the MSE loss function. If the number of MSE values is sufficient to determine a clear threshold, the process proceeds to identify the threshold that classifies data as either anomalous or normal. Conversely, if the number of MSE values is insufficient, the process returns to the data acquisition step and repeats the same processing steps until the required number of MSE values is obtained to establish the threshold for distinguishing normal and anomalous data.

Once the required number of MSE values is obtained, the gamma distribution [34] is applied to calculate the threshold based on the collected MSE values. The gamma distribution helps determine an accurate boundary, ensuring that MSE values below the threshold are classified as normal, while those exceeding the threshold are identified as anomalous [13]. To achieve this, we developed an additional gamma library implemented through the following steps.

The first step in this process involves using the data to estimate the parameters of the gamma distribution, specifically the shape and scale parameters denoted a and b, respectively. This is done by calculating the mean and the logarithmic mean of the collected MSE values. To estimate the parameter a, the mean $\bar{x}$ of the MSE data is first calculated using the formula:



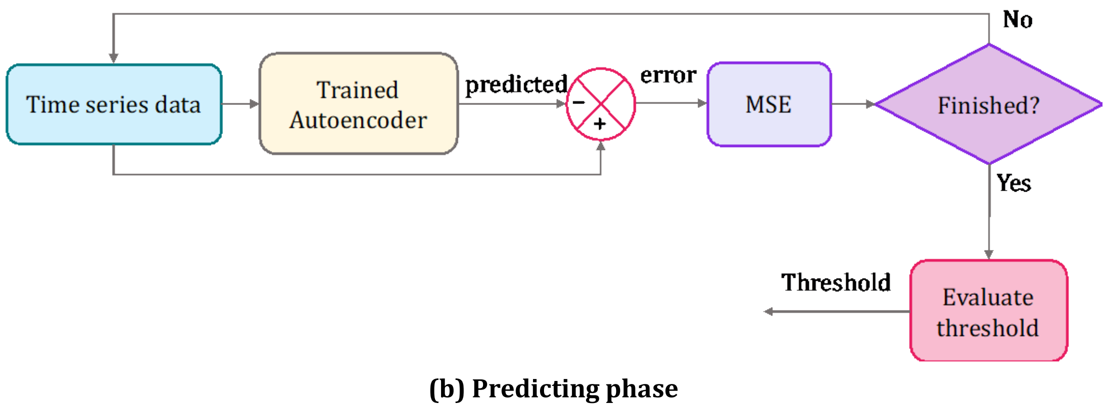


Fig. 4 *Two phases of anomaly detection design process*

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{7}$$

Next, the logarithmic mean of the data is calculated.

$$\log_{mean} = \frac{1}{n}\sum_{i=1}^{n} \log\,(x_i) \tag{8}$$

These two values provide an initial insight into the characteristics of the data and serve as the foundation for further calculations of the gamma distribution parameters. After obtaining the mean and logarithmic mean, the method of moments is used to estimate the initial value for the parameter a. This is calculated using the following formula:

$$s = \log(\bar{x}) - \log_{mean} \tag{9}$$

From this, the initial estimate for the parameter a is:

$$a_{est} = \frac{3 - s + \sqrt{(s-3)^2 + 24s}}{12s} \tag{10}$$

This is a quick approach to obtaining an initial value for a based on the difference between the mean and logarithmic mean. Next, the parameter a is further refined for greater accuracy using the Brent algorithm [35]. This algorithm is employed to solve nonlinear equations.

$$\log(a) - \psi(a) - s = 0 \tag{11}$$

Where $\psi(a)$ is the digamma function [36]. This enables the precise determination of parameter a, ensuring that it accurately reflects the characteristics of the dataset. Finally, after determining the shape parameter a, the scale parameter b can be calculated using the formula:

$$b = \frac{\bar{x}}{a} \tag{12}$$

The parameter b determines the scale of the gamma distribution based on the relationship between the mean of the dataset and the shape parameter a.

After estimating the parameters, a and b of the gamma distribution, the next step is to calculate the threshold for identifying values that can be considered anomalous. This is achieved using the PPF percent-point function (PPF) [37], also known as the inverse cumulative distribution function. The basic formula for calculating the threshold is as follows:

$$threshold = \Gamma^{-1}(a, q)b \tag{13}$$

Γ^{-1} is the inverse cumulative gamma function, where q represents the probability used to calculate the threshold. To determine the threshold value, the Newton-Raphson method [38] is applied to iteratively optimize the threshold through the following calculations:

$$x_{n+1} = x_n - \frac{I(a,\ x_n) - y}{f(x_n)} \tag{14}$$

The function $I(a, x_n)$ is the incomplete gamma function, and $f(x_n)$ is the derivative of the gamma probability density function. When x is smaller than $a + 1$, the series approximation for the lower incomplete gamma function [39] is used. When $x \geq a + 1$ continued fractions are applied to calculate the upper incomplete gamma function [40]. To ensure computational accuracy, we also employed the logarithmic approximation of the gamma function using the Lanczos method [41].

$$\log(\Gamma(x)) = \log\left(\sqrt{2\pi}\frac{x^{x-0.5}}{e^x}\sum_{i=0}^{6}\frac{c_i}{x+i}\right) \tag{15}$$

Where $\Gamma(x)$ is the gamma function, and c_i are the Lanczos coefficients (also known as Lanczos constants) which are carefully chosen to provide the best approximation of the gamma function. In the gamma library, we have developed a class called *gamma_gen*, which contains three important methods: *pdf*, *ppf*, and *fit*. Notably, the fit method plays a crucial role in estimating the two parameters, shape a and scale b. These parameters are then used in the *ppf* method to calculate the important thresholds of the gamma distribution. Algorithm 1 outlines the process for calculating the threshold.

Algorithm 1: Evaluate anomalous threshold

```
00  Input:
        // list of MSE values.
01      X → MSE values
02      $q$ → threshold probability
03  Step 1:
        // Find the parameters $a$ (shape) and $b$ (scale) of the gamma distribution from data X
04      $a, b \rightarrow fit(X)$
05  Step 2:
        // Calculate the threshold based on the threshold probability 1 - $q$ and the parameters $a, b$.
06      $threshold \rightarrow ppf(1 - q, a, b)$
07  Output:
        // cutoff value from gamma distribution.
08  threshold
```

To evaluate the effectiveness of the AE, the following metrics are used: *Accuracy*, *Precision*, *Recall*, and *F1-score*. *Accuracy* provides overall information about the correct classification rate, while *Precision* and *Recall* allow for a detailed assessment of the model's ability to identify anomalous samples. The *F1-score* is particularly useful when there is a significant imbalance between the classes, as it combines both *Precision* and *Recall*, offering a balanced measure of the model's performance. These metrics not only help assess accuracy but also ensure that the model can reliably detect anomalous samples in real-world environments, where sensitivity and precision are crucial [42]. The mathematical representations of these metrics are as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{16}$$

$$Precision = \frac{TP}{TP + FP} \tag{17}$$

$$Recall = \frac{TP}{TP + FN} \tag{18}$$

$$F1\_score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{19}$$

Where True Positive (TP) refers to samples that are labeled and predicted as anomalies, True Negative (TN) refers to samples that are labeled as normal but predicted as anomalous, False Positive (FP) refers to samples that are labeled as anomalies but predicted as normal, and False Negative (FN) refers to samples that are labeled and predicted as normal.

2.1.4 The Hardware Platform for Deploying Training and Inference of the AE Model

Fig. 5 illustrates the proposed hardware platform of this study. The processor part consists of two ESP32 microcontrollers, designed for real-time data acquisition, training, and inference using NeuronLite. In this system, ESP32-1 is responsible for capturing and processing analog signals via the analog-to-digital converter (ADC). The acquired data is then transmitted to ESP32-2 through UART1. ESP32-2 processes the received data and stores the necessary information on a secure digital (SD) card via the serial peripheral interface (SPI). At the same time, data intended for monitoring is sent to a computer through a USB2COM converter for display and analysis. The system also utilizes GPIO pins to optionally control the training or inference processes.

Figure 6 illustrates the algorithm of the anomaly detection system implemented on two ESP32 devices. First, ESP32-1 initializes the system and waits for a request from ESP32-2. Upon receiving the request, it collects time-series data, processes it, converts the data into an image, and then transmits the image to ESP32-2. On ESP32-2, after initialization, the system checks its operating mode. If the training mode is activated, ESP32-2 continuously receives image data from ESP32-1 to train the AE model. The training process terminates when one of two conditions is met: the training time exceeds the predefined limit or the loss value drops below the target threshold. Once the stopping condition is satisfied, the trained model is saved for use in the next phase. Next, the system enters the MSE threshold determination phase. In this step, ESP32-2 uses the trained AE model and continues receiving new images from ESP32-1. For each input image, the model performs reconstruction and calculates the MSE. This process repeats until a sufficient number of MSE samples are collected (e.g., 1000 values). From this dataset, the system computes and stores the optimal MSE threshold, completing the "Calculate and save threshold" step. After the threshold is determined, the system transitions to the anomaly detection phase. ESP32-2 continuously receives new images from ESP32-1, reconstructs each one using the AE model, and compares the resulting MSE to the predefined threshold. If the MSE is below the threshold, the data is identified as normal; otherwise, it is identified as abnormal, and the corresponding result is displayed. After processing, ESP32-2 checks whether retraining is required. If not, the system continues monitoring in prediction mode; if yes, it returns to the training phase to update the model.

If a pre-trained model is already available, ESP32-2 can skip the training phase and directly load the AE model along with the saved MSE threshold. In this case, the system operates solely in detection mode—continuously receiving data from ESP32-1, reconstructing images, computing MSE values, and evaluating operational status in real time.

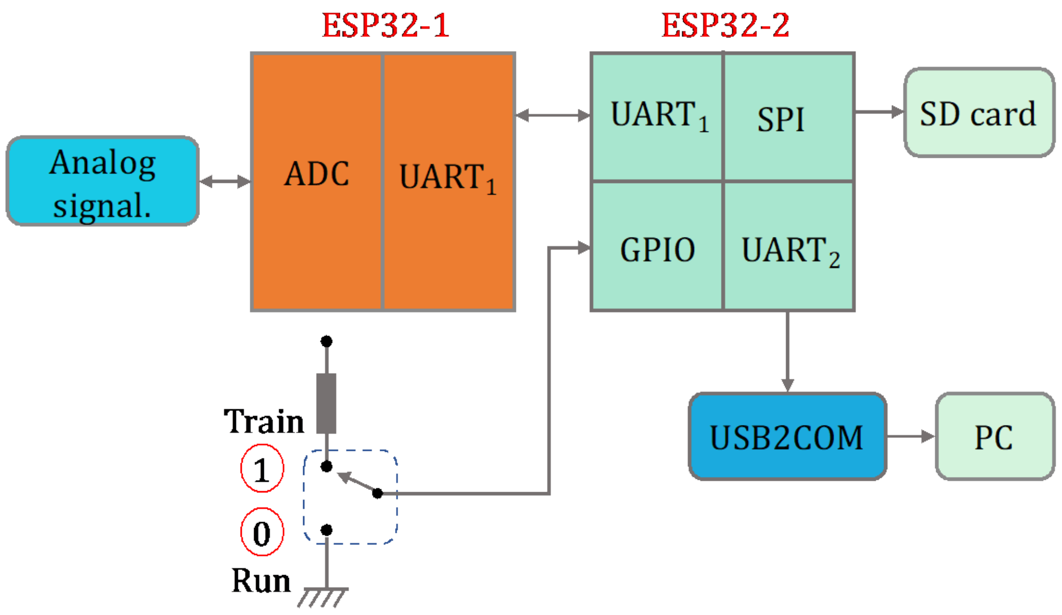


Fig. 5 *Hardware platform for testing the AD model based on NeuronLite*

2.2 Case Study: Abnormal Detection for Three-Phase Motor

In this study, the noise generated by a three-phase motor is acquired and converted into frequency spectrograms to support tasks such as model training, error analysis between normal motor operation, threshold determination for anomaly detection, and system testing with both normal and anomalous noise. The sliding window algorithm, as proposed in [43-45], is employed to generate continuous spectrograms for the system. Due to differing durations for training and prediction tasks, the algorithm must account for elapsed system time to update the spectrogram with new columns accordingly. The algorithm has been refined, as illustrated in Fig. 7, to align with the requirements of this research.

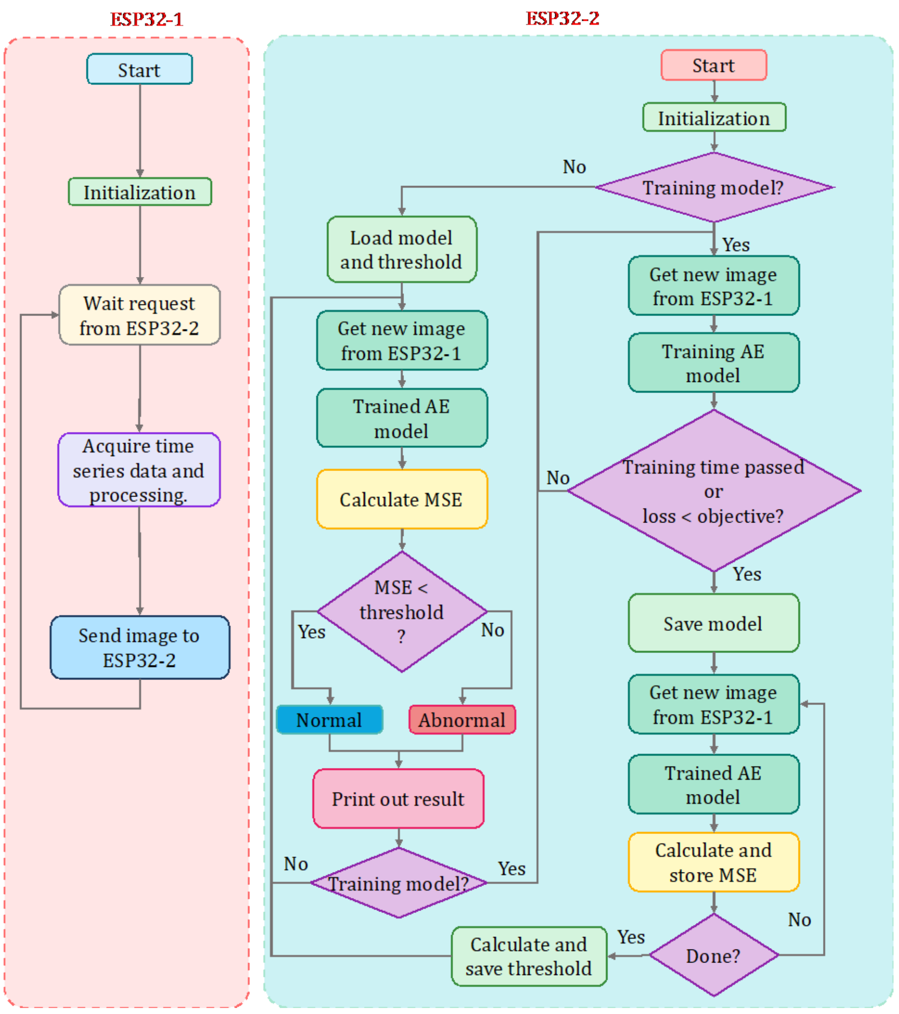


Fig. 6 *Algorithm of the two ESP32 microcontrollers for implementing the real-time training AD application*

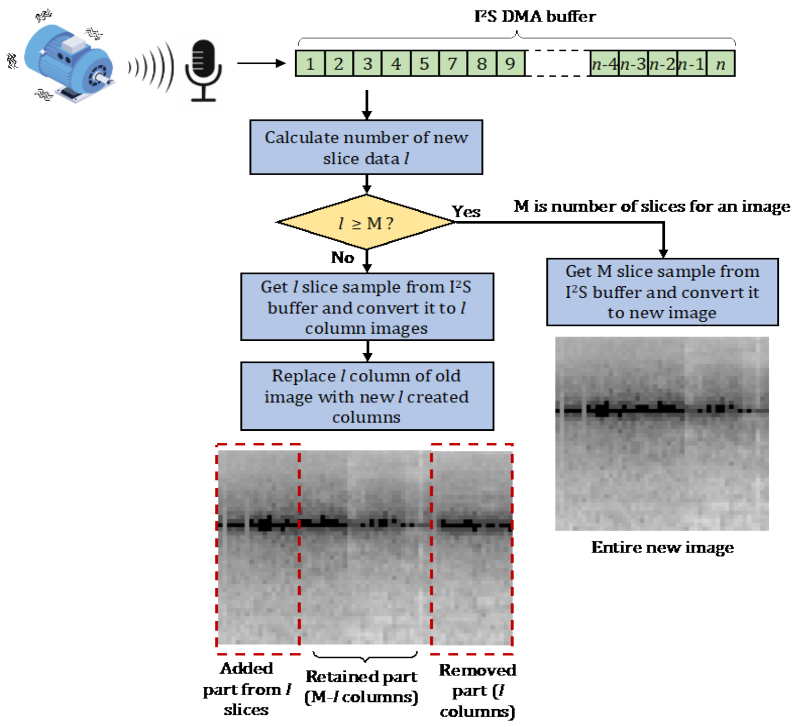

I²S DMA buffer
1 2 3 4 5 7 8 9 n-4 n-3 n-2 n-1 n
Calculate number of new slice data l
l ≥ M ?
Yes
No
M is number of slices for an image
Get l slice sample from I²S buffer and convert it to l column images
Replace l column of old image with new l created columns
Get M slice sample from I²S buffer and convert it to new image
Entire new image
Added part from l slices
Retained part (M-l columns)
Removed part (l columns)

Fig. 7 *Realtime Mel scale spectrogram generation algorithm*

The spectrograms utilized in this study are 48×48 in size, with the vertical axis representing frequency and the horizontal axis representing time. The maximum frequency on the vertical axis is 8 kHz, with each row corresponding to the average value of a Mel filter bank bin. The horizontal axis represents time, where 48 columns correspond to one second. For this setup, the sliding window has a width of 40 ms and slides by 20 ms per step. Within the program, the ESP32's I2S module is configured to continuously sample audio at a rate of *fs*=16kHz.

At each point when a spectrogram needs to be updated, the program calculates the number of slides *l* based on the elapsed time to determine the number of samples required to generate the new portion of the spectrogram. If *l* is smaller than the maximum number of slides *M*, *M* slides' worth of samples are acquired and converted into a new spectrogram. Otherwise, *l* slides' worth of samples is acquired and used to create l new columns for the previously generated spectrogram, with the *l* oldest columns being removed. The number of samples in *l* slides is calculated using the following formula:

$$l = \frac{time\_passed}{slide\_size} \tag{20}$$

$$number\_of\_sample = \frac{l \times f_s \times slide\_size}{1000} \tag{21}$$

The samples within a window are analyzed into 512 frequency components using the discrete Fourier transform (DFT) based on the formula proposed by [46], and only the first 256 components are utilized.

$$X_k = \sum\nolimits_{n=0}^{N-1} A(k) W^{jk}, j = 1, 2, 3, \quad \dots, N-1 \tag{22}$$

Where, *A(k)* is a complex number, and W is calculated using the following formula:

$$W = e^{2\pi i/N} \tag{23}$$

The power of each frame is calculated using the formula:

$$P_i(k) = \frac{1}{N} |X_k|^2 \tag{24}$$

Next, the Mel filter bank is calculated to group frequency bands into *n* groups. The formula for converting the frequency scale from Hertz to the Mel scale is presented as follows [47]:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) = 1127\ln\,(1 + \frac{f}{700}) \tag{25}$$

And the inverse transformation is given by the formula:

$$f = 700\left(10^{\frac{m}{2595}} - 1\right) = 700(e^{\frac{m}{1127}} - 1\,) \tag{26}$$

The frequencies f_{high} and f_{low} are converted to the Mel scale to determine the upper M_{high} and lower M_{low} bounds of the filter bank, *m* linearly spaced values within the range $[M_{low};\ M_{high}]$ are calculated and converted back to the frequency scale using the inverse transformation formula. Subsequently, the filters are generated using the following formula:

$$H_m(k) = \begin{cases} 0 \quad , & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)}, & f(m-1) \leq k \leq f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)}, & f(m) \leq k \leq f(m+1) \\ 0 \quad , & k > f(m+1) \end{cases} \tag{27}$$

With *m* being the desired number of filters and *f(.)* representing the list of frequencies spaced by *m*+2 Mel.

After converting the spectrogram images, the resulting data ranges from -128 to 127. To normalize the data to the [0,1] range and enhance the learning efficiency of the autoencoder model, we applied equation. (27):

$$x_{scale} = \frac{x + 128}{255} \tag{28}$$

After data collection, the full dataset is input into the AE model (Fig. 4) for training, using the previously described algorithms and hardware. To evaluate training time, prediction capability, and performance, six models were configured with different learning rates and optimizers. Training was based on two criteria: loss value and training time. Table 1 details the configurations. models 1 and 4 use Adam with a 0.001 learning rate, evaluated by loss (model 1) and time (model 4) to test Adam's training acceleration. The other models use SGD with learning rates of 0.01 (models 2, 5) and 0.1 (models 3, 6); models 2 and 3 are evaluated by loss, and models 5 and 6 by time. A 0.001 learning rate was excluded for SGD due to long training times. This setup allows comparison of optimization performance and suitability for real-world demands.

Table 1 *Configuration of the models deployed for real-world testing*

	Learning rate	Optimizer	Criteria
Model 1	0.001	Adam	Loss
Model 2	0.01	SGD	Loss
Model 3	0.1	SGD	Loss
Model 4	0.001	Adam	Time
Model 5	0.01	SGD	Time
Model 6	0.1	SGD	Time

2.3 Experimental Setup

Fig. 8 illustrates the experimental setup, where the anomaly detection model is placed near the three-phase motor. The model is trained and tested using noise emitted by the motor. Initially, it is trained with noise from normal operation. After training, it continues to run with normal noise to determine the anomaly detection threshold. Then, faults such as phase loss, phase shift, and bearing failure are introduced to test the model's detection capability.

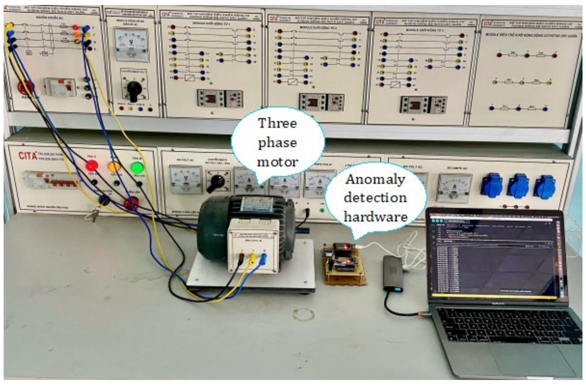


Fig. 8 *Layout of the training and testing experiment for the anomaly detection model*

3. Experimental Results

3.1 Training Results

For the loss criterion, the training process will automatically stop when the loss value drops below 0.04. This target should be adjusted based on the object to ensure the quality of the learning process. The target loss value is calculated based on the overall mean of the MSE values throughout the training process. This approach helps the model learn stably and achieve better recognition performance. Once the target loss value is reached, all weights, including the weight and bias, are saved to the SD card for reloading when necessary. Models 1, 2, and 3 are trained based on the target loss value. Table 2 presents the training time results for these models.

The training times for these models vary significantly, reflecting the acceleration effectiveness of the optimizers. The results show that the SGD algorithm is less effective in accelerating training. Specifically, models 2 and 3, which apply this algorithm, require training times of 79.4 and 7.45 minutes, respectively. Model 3 has a faster training time because the learning rate is relatively large at 0.1. When the learning rate is reduced by a factor of 10 to 0.01 (model 2), the training time increases to 79.4 minutes, which is more than 10 times longer. Meanwhile, model 1, with a learning rate of 0.001 (100 times smaller than model 2), requires only 9.39 minutes to reach the training goal. This indicates that the Adam algorithm optimizes the training process well. However, due to its fast convergence, the model may not have been trained with all the features of the monitored object. This could be considered a disadvantage of target-based training in the anomaly detection problem.

For the time criterion, models 4, 5, and 6 were trained for a duration of 30 minutes. The training time should be adjusted based on the object being monitored, ensuring it is long enough for the models to learn all the necessary features of the object. Fig. 9 shows the change in loss values for the three models over the 30-minute training period. This graph indicates that models 4 and 6 converge very quickly, with the loss value decreasing to around 0.05 within the first 10 minutes and continuing to decrease steadily over time. In contrast, Model 5 decreases its loss much more slowly and maintains a higher loss value compared to the other two models. This is understandable because model 5 is trained with a learning rate of 0.01, resulting in slower weight updates. It also demonstrates that the SGD algorithm is less effective than Adam. The training results of models 4 and 6 highlight the efficiency of the Adam algorithm, as it enables model 4, with a learning rate of 0.001, to achieve a training speed comparable to that of model 6, which uses a learning rate of 0.1.

Table 2 *Training time with target loss*

	Model 1	Model 2	Model 3
Times (minute)	9.39	79.40	7.45

3.2 Abnormal Detection Results in the Working Noise of Three-Phase Motor

After training, the AE models are used for inference on both normal and abnormal motor noise. Three faults—phase loss, phase shift, and bearing failure—were deliberately introduced. For each case, the models processed 1000 noise samples to generate 1000 MSE values. MSE values from normal noise are used to set the anomaly detection threshold via the gamma distribution. MSE values from abnormal cases are used to assess the model's detection accuracy.

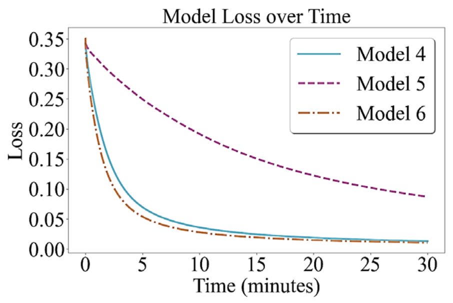


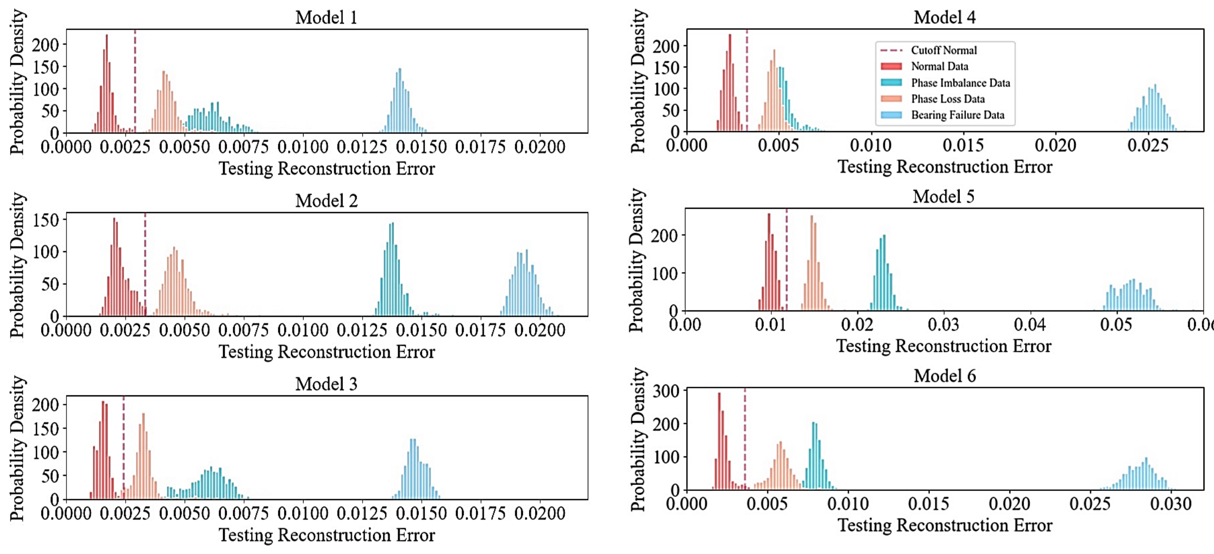
Fig. 9 *Training process of Model 4, Model 5 and Model 6*

Fig. 10 (a) shows the MSE distribution of Models 1, 2, and 3 for normal and abnormal motor noise (phase shift, phase loss, bearing failure). All models effectively separate normal and abnormal data. Bearing failure has the most distinct MSE distribution, making it the easiest to detect. However, phase shift and phase loss in Models 1 and 3 are closer to the normal range, increasing the chance of confusion. In contrast, Model 2 only shows phase loss data overlapping with normal.

The anomaly threshold is determined based on a probability threshold of 0.01%, meaning that any noise segment with an MSE probability below 0.01% in the normal distribution is considered abnormal. Thus, when training the model with a target loss, the models exhibit different distributions and are all capable of detecting anomalies effectively in the working noise of the three-phase motor. The anomaly thresholds for Models 1, 2, and 3, determined using the gamma distribution, are 0.0029, 0.0033, and 0.0024, respectively.

Similarly, the graphs in Fig. 10 (b) show the MSE distribution of Models 4, 5, and 6 when operating with normal, phase-shifted, phase-lost, and bearing-failure motor noise. These distributions clearly illustrate the difference between the MSE distribution of normal motor operation and the abnormal cases, similar to the models trained with the target loss. However, the distributions show a noticeable gap between the normal and abnormal distributions, meaning that if the anomaly threshold is well defined, the models will detect anomalies with high accuracy and minimal confusion. This could be evidence that the models were trained with most of the normal noise characteristics of the three-phase motor. Unlike the target loss criterion, here the anomaly threshold is determined based on a probability threshold of 0.001%, meaning that any noise segment with an MSE probability below 0.001% in the normal distribution is considered abnormal. The anomaly thresholds for Models 4, 5, and 6, determined using the gamma distribution, are 0.0033, 0.0118, and 0.0036, respectively.

In summary, the anomaly detection model for the working noise of a three-phase motor has been successfully developed based on the NeuronLite library created by the authors. The anomaly detection model can be trained in real-time while the motor is operating. The training can be based on either the target loss value or training time, depending on the object being monitored. In the experiment conducted in this study, both training solutions performed well in classifying normal and abnormal motor noise. The accuracy of anomaly detection will be analyzed in detail in the next section.



(a) Models trained with target loss **(b) Models trained with specific time duration**

Fig. 10 *Realtime testing results of models*

Fig. 11 presents the confusion matrix of the six models. Each matrix shows the number of correct and incorrect classifications for the two types of data: normal and abnormal. Fig. 11 (a) shows that model 1 correctly classifies 963 normal samples and 2999 abnormal samples, but it misclassifies 37 normal samples as abnormal and 1 abnormal sample as normal. Fig. 11 (b) shows that model 2 correctly classifies 985 normal samples and 3000 abnormal samples, with only 15 normal samples misclassified as abnormal and no misclassification of abnormal samples. In Fig. 11 (c), the confusion matrix of model 3 shows 980 normal samples and 2975 abnormal samples correctly classified, with 20 normal samples misclassified as abnormal and 25 abnormal samples misclassified as normal. Fig. 11 (d) and 11 (e) show that models 4 and 5 achieve perfect performance, correctly classifying all 1000 normal samples and 3000 abnormal samples without any misclassifications. Fig. 11 (f) shows that model 6 correctly classifies 984 normal samples and 3000 abnormal samples, with only 16 normal samples misclassified as abnormal and no misclassification of abnormal samples. In summary, models 4 and 5 demonstrate the highest accuracy, classifying all samples correctly. Models 2 and 6 also perform well, with minimal misclassifications. Models 1 and 3 show a higher misclassification rate but still achieve relatively good performance in classifying normal and abnormal data.

These results suggest that training based on time yields better anomaly detection performance, as the models trained with the target loss may have learned the full features of the normal motor noise. On the other hand, when models are trained quickly (Models 1 and 3), they likely miss many of the normal noise features, which may explain the higher misclassification rate in these models. Meanwhile, model 2, which was trained for a longer duration (over 79 minutes), has a lower misclassification rate. The models trained based on time perform almost perfectly. The difference in accuracy between model 2 (trained for over 79 minutes) and model 5 (trained for 30 minutes) could be due to different experimental conditions, as this is a real-time experiment that may be affected by surrounding noise.

Table 3 shows the accuracy of the six models. The results indicate that models 4 and 5 achieved absolute performance, with all metrics at 100%, demonstrating their perfect classification capability. Models 2 and 6 also performed very well, with accuracies of 99.62% and 99.6%, respectively, and Precision, Recall, and F1-scores all very close to 100%, indicating that these models effectively detected abnormal noises. Models 1 and 3 performed slightly lower, with accuracies of 99.05% and 98.87%, respectively. However, both still achieved good results, with an F1-score of 99.37% for model 1 and 99.24% for model 3, reflecting a balanced ability to detect and correctly predict anomalous cases.

Table 3 *Accuracy of six models*

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
Accuracy	99.05	99.62	98.87	100.0	100.0	99.60
Precision	98.78	99.5	99.33	100.0	100.0	99.46
Recall	99.96	100.0	99.16	100.0	100.0	100.0
F1-score	99.37	99.75	99.24	100.0	100.0	99.73

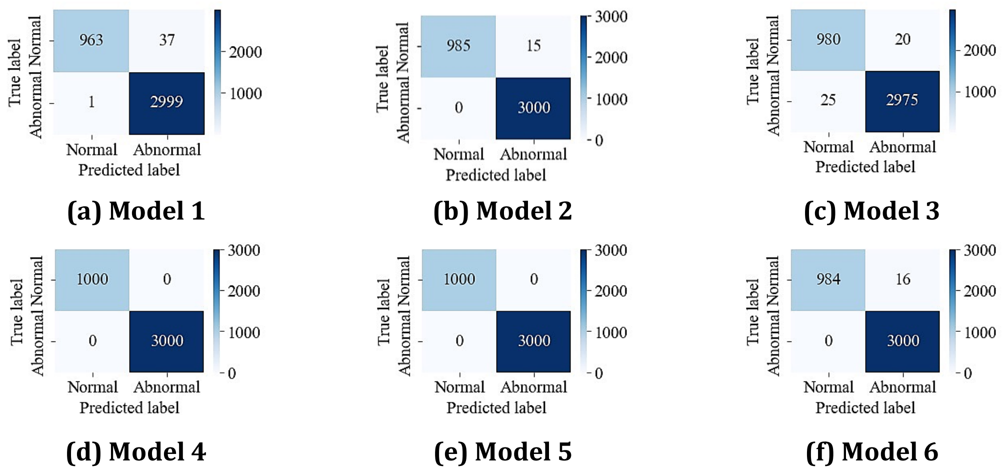


Fig. 11 *Confusion matrix of 6 models*

The metrics indicate that all six models effectively detected and classified normal and abnormal data, with models 4 and 5 performing best. High Precision and Recall values confirm the models' stability and reliability in real-world settings. These results validate the accuracy of the real-time training algorithm integrated into NeuronLite and the effectiveness of the optimization methods. This highlights the potential for applying this approach to unsupervised or semi-supervised learning on embedded systems.

3.3 Evaluate the computation time of the models

Table 4 compares the training time for each model on an MCU and a PC. On the MCU, the Adam algorithm takes about 531 ms per training session, which is approximately 7.5 times slower than the SGD algorithm. When using the NeuronLite library implemented in C/C++ on a PC, the training time for Adam and SGD reduces to 2.57 ms and 1.56 ms, respectively. This is understandable because the PC has a higher processing speed and more efficient floating-point computation capabilities. The results show that a single training session using the SGD optimizer is significantly faster than Adam. However, the Adam algorithm helps the AE model converge more quickly, leading to faster training time, as noted earlier. Additionally, the time required to find the anomaly detection threshold is also part of the training process. This time is only 9 ms when performed with 1000 MSE samples. This time is relatively small and does not significantly affect the overall training time or the time required to determine the threshold for the anomaly detection model.

Table 4 *Training time (ms) per spectrogram image*

	MCU	PC
Adam	531.2	2.57
SGD	70.5	1.56

The time required to encode an input of the trained AE model to find the MSE is also crucial in demonstrating the real-time operational capability of the anomaly detection model. The encoding time for a spectrogram image of the trained AE model was measured on both the MCU and PC platforms. The results show that the MCU takes 29.95 ms to encode one image, while the PC takes only 0.2 ms, which is approximately 150 times faster than the MCU. However, the 30 ms encoding time on the embedded system is still sufficient for real-time applications.

4. Discussion

This study has successfully developed a real-time training-based AD system entirely running on the embedded system. Currently, there is a lack of studies that implement real-time training mechanisms for the AD tasks on MCUs, with most research using FPGA platforms as outlined in the Introduction. Therefore, the performance of this study will be compared with a similar model implemented on FPGA from the study [18]. Study [18] is chosen for comparison because it fully reports training and prediction latency. Table 5 compares the hardware platform and performance of the anomaly detection task between this study and [18]. Study [18] uses an AI ONLAD core running on the PYNQ-Z1 FPGA (clocked at 100 Hz) with a model configuration of 512 input nodes, 16 hidden nodes, and 512 output nodes. In this case, the AE model has 16,400 parameters, with a training time of 0.14 ms and a prediction time of 0.1 ms—ideal for real-time tasks. In contrast, NeuronLite was only tested on the ESP32 MCU (clocked at 240 MHz), which leads to larger training times (138.8 ms and 17.6 ms for the Adam and SGD optimizers, respectively) and a prediction time of 29.5 ms. Although these results are not as impressive, the MCU-based system is compact, development procedures are simpler, power consumption is low, and the cost is cheaper, making it very suitable for mobile applications and IoT systems requiring embedded intelligence. While FPGA offers higher speed and accuracy, its high cost and complex programming limit its widespread deployment.

Table 5 *Comparison of results between other studies*

Name papers	[18]	This study
Hardware platform	PYNQ-Z1	2
Frequency (MHz)	100	4
Problem type	Anomaly Detection	Abnormal Detection
Number of parameters	16,400	2,882
Training latency (ms)	~ 0.14	Adam = 132.8 and SGD = 17.6
Predict latency (ms)	~ 0.1	7.4

The AE-based anomaly detection models and NeuronLite library have strong development potential. Real-time training with optimizers like Adam enables applications in industrial monitoring, medical diagnostics, and fault detection. Compatibility with MCUs allows integration into IoT systems with embedded intelligence. Future directions include adding more optimization algorithms, improving computational performance, addressing concept drift to enhance model robustness, and integrating supervised learning to broaden NeuronLite's application scope.

5. Conclusion

The study has successfully developed a comprehensive anomaly detection platform based on the NeuronLite library created by the authors. The developed platform is capable of automating the entire process, from training and threshold determination to anomaly detection, on the proposed hardware platform. Notably, the proposed algorithm allows the AE model to be trained based on either the target loss or training time, enabling the system to effectively learn the normal features of various objects, from simple to complex. The proposed anomaly detection models were applied to detect anomalies in the noise of a three-phase motor. The experimental results show that, although the Adam optimization algorithm requires more computational resources, it provides faster convergence, thereby effectively learning the object's features. The gamma distribution was successfully applied to determine the anomaly threshold directly on the system, yielding the best anomaly detection result, achieving 100% accuracy under the experimental conditions of the study. Future development directions include expanding the optimization algorithms, addressing the "concept drift" problem, and integrating semi-supervised learning capabilities to enhance the adaptability and broaden the application of NeuronLite. Displayed equations should be numbered consecutively in each section, with the number set flush right and enclosed in parentheses.

Acknowledgement

The authors used the free version of QuillBot to assist in grammar checking and language refinement. All content generated was reviewed and verified by the authors, who take full responsibility for the final submission.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design****: Bao-Toan Thai and Van-Khanh Nguyen;* ***data collection****: Bao-Toan Thai and Vy-Khang Tran;* ***analysis and interpretation of results****: Bao-Toan Thai, Vy-Khang Tran, Chi-Ngon Nguyen and Van-Khanh Nguyen;* ***draft manuscript preparation****: Bao-Toan Thai and Van-Khanh Nguyen. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, *41*(3), 1–58. https://doi.org/10.1145/1541880.1541882

[2] Tabassum, M., Mahmood, S., Bukhari, A., Alshemaimri, B., Daud, A., & Khalique, F. (2024). Anomaly-based threat detection in smart health using machine learning. *BMC Medical Informatics and Decision Making*, *24*(1), 347. https://doi.org/10.1186/s12911-024-02760-4

[3] Akinade, S. K. (2024). Implementing AI-driven anomaly detection for cyber-security in healthcare networks. *ATBU Journal of Science, Technology and Education*, *12*(2), 598–610.

[4] Grunova, D., Bakratsi, V., Vrochidou, E., & Papakostas, G. A. (2024). Machine learning for anomaly detection in industrial environments. *Engineering Proceedings*, *70*(1), 25. https://doi.org/10.3390/engproc2024070025

[5] Gribbestad, M., Hassan, M. U., Hameed, I. A., & Sundli, K. (2021). Health monitoring of air compressors using reconstruction-based deep learning for anomaly detection with increased transparency. *Entropy*, *23*(1), 83. https://doi.org/10.3390/e23010083

[6] Cowton, J., Kyriazakis, I., Plötz, T., & Bacardit, J. (2018). A combined deep learning GRU-autoencoder for the early detection of respiratory disease in pigs using multiple environmental sensors. *Sensors*, *18*(8), 2521. https://doi.org/10.3390/s18082521

[7] Zhang, H., Guo, W., Zhang, S., Lu, H., & Zhao, X. (2022). Unsupervised deep anomaly detection for medical images using an improved adversarial autoencoder. *Journal of Digital Imaging*, *35*(2), 153–161. https://doi.org/10.1007/s10278-021-00558-8

[8] Sato, D., et al. (2018). A primitive study on unsupervised anomaly detection with an autoencoder in emergency head CT volumes. In *Medical Imaging 2018: Computer-Aided Diagnosis* (Vol. 10575, pp. 388–393). SPIE. https://doi.org/10.1117/12.2292276

[9] Akcay, S., Atapour-Abarghouei, A., & Breckon, T. P. (2019). Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Computer Vision – ACCV 2018* (pp. 622–637). Springer. https://doi.org/10.1007/978-3-030-20893-6_39

[10] Akçay, S., Atapour-Abarghouei, A., & Breckon, T. P. (2019). Skip-ganomaly: Skip connected and adversarially trained encoder-decoder anomaly detection. In *2019 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). IEEE. https://doi.org/10.1109/IJCNN.2019.8851808

[11] Hammad, S. S., Iskandaryan, D., & Trilles, S. (2023). An unsupervised TinyML approach applied to the detection of urban noise anomalies under the smart cities environment. *Internet of Things*, *23*, 100848. https://doi.org/10.1016/j.iot.2023.100848

[12] Charan, K. S. (2022). An auto-encoder based TinyML approach for real-time anomaly detection. *SAE International Journal of Advances and Current Practices in Mobility*, *5*, 1496–1501. https://doi.org/10.4271/2022-28-0406

[13] Nguyen, V.-K., Thai, B.-T., Tran, V.-K., Pham, H., & Nguyen, C.-N. (2024). Real-time anomaly detection in electric motor operation noise. *IAES International Journal of Artificial Intelligence (IJ-AI)*, *13*(4), 13. https://doi.org/10.11591/ijai.v13.i4.pp3814-3826

[14] Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of the ICML Workshop on Unsupervised and Transfer Learning* (pp. 37–49).

[15] Akin, E., Aydin, I., & Karakose, M. (2011). FPGA-based intelligent condition monitoring of induction motors: Detection, diagnosis, and prognosis. In *2011 IEEE International Conference on Industrial Technology* (pp. 373–378). IEEE. https://doi.org/10.1109/ICIT.2011.5754405

[16] Wess, M., Manoj, P. S., & Jantsch, A. (2017). Neural network-based ECG anomaly detection on FPGA and trade-off analysis. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1–4). IEEE. https://doi.org/10.1109/ISCAS.2017.8050805

[17] Moss, D. J., Boland, D., Pourbeik, P., & Leong, P. H. (2018). Real-time FPGA-based anomaly detection for radio frequency signals. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1–5). IEEE. https://doi.org/10.1109/ISCAS.2018.8350890

[18] Tsukada, M., Kondo, M., & Matsutani, H. (2020). A neural network-based on-device learning anomaly detector for edge devices. *IEEE Transactions on Computers*, *69*(7), 1027–1044.

[19] Decherchi, S., Gastaldo, P., Leoncini, A., & Zunino, R. (2012). Efficient digital implementation of extreme learning machines for classification. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *59*(8), 496–500. https://doi.org/10.1109/TCSII.2012.2204112

[20] Yeam, T. C., Ismail, N., Mashiko, K., & Matsuzaki, T. (2017). FPGA implementation of extreme learning machine system for classification. In *TENCON 2017 – IEEE Region 10 Conference* (pp. 1868–1873). IEEE. https://doi.org/10.1109/TENCON.2017.8228163

[21] Frances-Villora, J. V., et al. (2016). Hardware implementation of real-time extreme learning machine in FPGA: Analysis of precision, resource occupation and performance. *Computers & Electrical Engineering*, *51*, 139–156. https://doi.org/10.1016/j.compeleceng.2016.02.007

[22] Basu, A., et al. (2013). Silicon spiking neurons for hardware implementation of extreme learning machines. *Neurocomputing*, *102*, 125–134. https://doi.org/10.1016/j.neucom.2012.01.042

[23] Frances-Villora, J. V., et al. (2018). Moving learning machine towards fast real-time applications: A high-speed FPGA-based implementation of the OS-ELM training algorithm. *Electronics*, *7*(11), 308. https://doi.org/10.3390/electronics7110308

[24] Safaei, A., et al. (2018). System-on-a-chip (SoC)-based hardware acceleration for an online sequential extreme learning machine (OS-ELM). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *38*(11), 2127–2138. https://doi.org/10.1109/TCAD.2018.2878162

[25] Wang, Z., et al. (2020). Computer-aided diagnosis based on extreme learning machine: A review. *IEEE Access*, *8*, 141657–141673. https://doi.org/10.1109/ACCESS.2020.3012093

[26] Chuang, Y.-C., et al. (2021). An arbitrarily reconfigurable extreme learning machine inference engine for robust ECG anomaly detection. *IEEE Open Journal of Circuits and Systems*, *2*, 196–209. https://doi.org/10.1109/OJCAS.2020.3039993

[27] Rout, S. K., & Biswal, P. K. (2020). Digital implementation of OS-ELM for data classification in real time. In *Advances in Electrical Control and Signal Systems* (pp. 339–347). Springer. https://doi.org/10.1007/978-981-15-5262-5_24

[28] Thai, B.-T., et al. (2024). On-device training of artificial intelligence models on microcontrollers. *IAES International Journal of Artificial Intelligence (IJ-AI)*, *13*(3), 11. https://doi.org/10.11591/ijai.v13.i3.pp2829-2839

[29] Mukkamala, M. C., & Hein, M. (2017). Variants of RMSProp and AdaGrad with logarithmic regret bounds. In *International Conference on Machine Learning* (pp. 2545–2553). PMLR. https://doi.org/10.48550/arXiv.1706.05507

[30] Liu, Y., Gao, Y., & Yin, W. (2020). An improved analysis of stochastic gradient descent with momentum. *Advances in Neural Information Processing Systems*, *33*, 18261–18271. https://doi.org/10.48550/arXiv.2007.07989

[31] Kingma, D. P. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. https://doi.org/10.48550/arXiv.1412.6980

[32] Zhang, Y. (2018). A better autoencoder for image: Convolutional autoencoder. In *ICONIP17-DCEC*.

[33] Pratella, D., et al. (2021). A survey of autoencoder algorithms to pave the diagnosis of rare diseases. *International Journal of Molecular Sciences*, *22*(19), 10891. https://doi.org/10.3390/ijms221910891

[34] Gordon, L. (1994). A stochastic approach to the gamma function. *The American Mathematical Monthly*, *101*(9), 858–865. https://doi.org/10.1080/00029890.1994.11997039

[35] Brent, R. P. (2013). *Algorithms for minimization without derivatives*. Courier Corporation.

[36] Chen, C.-P., Elezović, N., & Vukšić, L. (2013). Asymptotic formulae associated with the Wallis power function and digamma function. *Journal of Classical Analysis*, *2*(2), 151–166. https://doi.org/10.7153/jca-02-13

[37] Even, A., Shankaranarayanan, G., & Berger, P. (2008). Inequality in the utility of data: Modeling, assessment, and implications. In *Proceedings of the International Conference on Industrial Logistics* (pp. 1–9).

[38] Ypma, T. J. (1995). Historical development of the Newton–Raphson method. *SIAM Review*, *37*(4), 531–551. https://doi.org/10.1137/1037125

[39] Blahak, U. (2010). Efficient approximation of the incomplete gamma function for use in cloud model applications. *Geoscientific Model Development*, *3*(2), 329–336. https://doi.org/10.5194/gmd-3-329-2010

[40] DeFranco, M. (2019). On a series for the upper incomplete gamma function. *arXiv preprint arXiv:1909.06941*. https://doi.org/10.48550/arXiv.1909.06941

[41] Lanczos, C. (1950). An iteration method for the solution of the eigenvalue problem of linear differential and integral operators.

[42] Yacouby, R., & Axman, D. (2020). Probabilistic extension of precision, recall, and F1 score for more thorough evaluation of classification models. In *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems* (pp. 79–91). https://doi.org/10.18653/v1/2020.eval4nlp-1.9

[43] Nguyen, V. K., et al. (2022). Realtime non-invasive fault diagnosis of three-phase induction motor. *Journal of Technical Education Science*, *17*(Special Issue 03), 1–11. https://doi.org/10.54644/jte.72B.2022.1231

[44] Tran, V.-K., et al. (2024). A proposed approach to utilizing ESP32 microcontroller for data acquisition. *Journal of Engineering and Technological Sciences*, *56*(4), 474–488. https://doi.org/10.5614/j.eng.technol.sci.2024.56.4.4

[45] Warden, P., & Situnayake, D. (2019). *TinyML: Machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*. O'Reilly Media.

[46] Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, *19*(90), 297–301. https://doi.org/10.2307/2003354

[47] O'Shaughnessy, D. (1987). *Speech communications: Human and machine (IEEE)*. Universities Pres