

Intelligent Analysis of Sensor Data Based on SBCs to Monitor the Operating Status of Devices

Bartosz Dominikowski

Abstract—This article presents an advanced computing system integrating a microprocessor and a microcontroller to analyze signals from measuring transducers. AI-based intelligent algorithms were implemented to classify the operational state of the device. The applied architecture is based on the Debian operating system (on the microprocessor side) and the Zephyr RTOS real-time operating system (on the microcontroller side). As part of the research, various intelligent models were tested, analyzing their limitations and the level of implementation difficulty. The use of the Python language and a Colab cloud environment enabled a transparent creation of the model structure and their efficient integration with the proposed hardware system.

Keywords—SBC (Single Board Computer); AI (Artificial Intelligence); Classification; MLP (Multilayer Perceptron); Sensors

I. INTRODUCTION

THE main objective of this article is to present the possibilities of an easy implementation of various AI (Artificial Intelligence) algorithms supporting decision-making in SBC systems. The rapid advancement of digital devices now allows complex computational models to run locally on the device, eliminating the need for external server resources.

Continuous monitoring of operational parameters, such as nominal values, plays a key role in ensuring the functional safety of electrical, electronic, and programmable electronic systems. This solution allows for the early detection of anomalies and the effective prevention of failures that could pose a threat to people or the environment. Thanks to the use of AI algorithms, it is possible to implement intelligent diagnostics that minimize downtime and optimize the performance of the monitored device.

This article presents the application of a hybrid data processing system consisting of an MPU (Microprocessor Unit) - QRB2210 (ARM Cortex-A53) and an MCU (Microcontroller Unit) - STM32U585 (ARM Cortex-M33), which utilizes algorithms to analyze the operational state of an object based on the load current waveform and data from measuring transducers such as: temperature, light intensity, vibrations, and instantaneous sound amplitude. The analysis of this measurement data is performed by an algorithm based on artificial intelligence. The proposed algorithm enables the detection of the device's operational states: normal and failure. The use of artificial intelligence in the proposed system allows the replacement of a signal analysis expert by an algorithm.

In the proposed System-on-Chip (SoC) microprocessor system, computational performance is based on the ARM (Advanced RISC Machine) architecture with the NEON extension. Thanks to SIMD (Single Instruction, Multiple Data) technology, the processor can process entire packets of float32 data simultaneously, which significantly accelerates floating-point operations. A detailed description of the proposed SBC system is given in [1]. A description of systems based on the ARM architecture is provided in [2]. The microprocessor (MPU) in the SBC (Single Board Computer) system operates under the control of the Linux Debian operating system, which is capable of handling complex AI computations. The microcontroller (MCU) uses the Zephyr RTOS (Real-Time Operating System), which enables multitasking. The current value from the tested object is acquired by using an electronically assisted current transformer and is transmitted to the analog input of the microcontroller, which uses its built-in ADC (Analog-to-Digital Converter) with a configurable resolution. These data are then stored in an array of `uint16_t` (unsigned 16-bit integer) and converted into binary data (bytes) before transmission.

The Linux OS (Operating System) and the microcontroller communicate through Bridge, Arduino's RPC (Remote Procedure Call) library. To determine the operational state of the tested device, signals from the Arduino Nano 33 BLE Sense Rev2 module were used. It has an integrated IMU (Inertial Measurement Unit), including an accelerometer, gyroscope, and magnetometer, as well as sensors for temperature, light intensity, and a digital microphone. The acquired information is transferred to AI algorithms for intelligent analysis. The reading of data from the built-in sensors is handled by the nRF52840 microcontroller, which transmits them to the Arduino UNO Q unit via the I²C bus. Then, using the RouterBridge library, these signals are sent from the MCU to the Linux/Python environment (MPU).

A key problem in maintaining the operational fluency of the SBC system is the need to minimize latencies (delays) in the exchange of information and commands between the MCU and MPU systems. The total duration of this process mainly consists of: ADC conversion, saving to RAM, and the transmission itself. In order to shorten the communication time, the `Bridge.notify()` function was used. Opposite the blocking `Bridge.call()` method, this function sends data without waiting for a response, which prevents halting the program execution until a result is received from Python.

Fig. 1 shows the schematic diagram of the module (Arduino UNO Q) used in the algorithm performance tests.



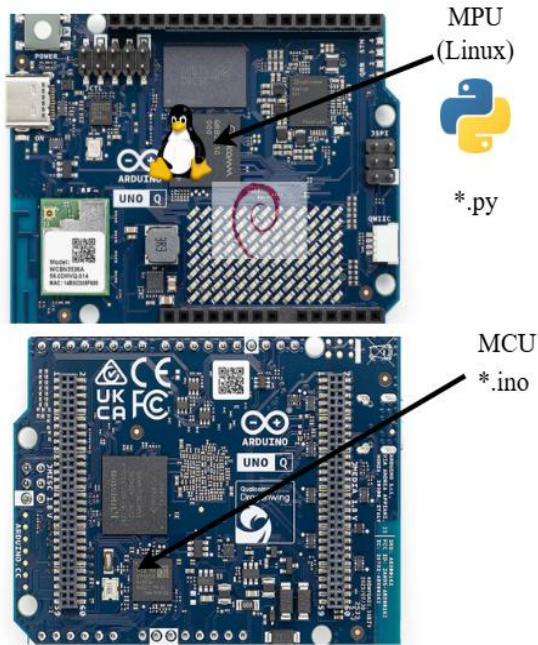


Fig. 1. The proposed SBC system: top side and bottom side.

The Arduino UNO Q module, shown in Figure 1, implements algorithms for testing the operating state of the device. The algorithms analyzing the measurement data were written in the Python programming language on the MPU microprocessor unit (Fig. 1 - *.py). The advantage of the proposed SBC system is the possibility to freely modify and optimize the algorithms thanks to full access to the source code. The use of the joblib library enables efficient serialization and management of full-scale models and classifiers. However, whether such models can be deployed on an MPU depends on the computational constraints of the target device, not on the serialization method itself. The use of the Python language in the Linux operating system allows for the deployment of ready-made AI models (e.g., from the following libraries: Scipy, NumPy, scikit-learn, TensorFlow/ Keras, ONNX, ONNX Runtime, skl2onnx) which shortens the code-writing time compared to low-level programming languages.

Arduino App Lab is a complete programming environment for the proposed SBC system, extensive enough that the development of the entire project is practically concentrated within a single tool. Fig. 2 shows the panel of this environment.

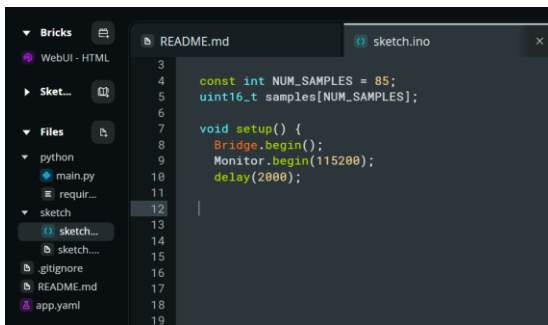


Fig. 2. Arduino App Lab development environment panel.

App Lab connects to the Arduino UNO Q module in the local network using mDNS (Multicast DNS), which allows for automatic device detection. The test program is created and run from the computer, and then the base code is transferred and saved in the SBC system. The main components of the programming environment shown in Fig. 2 are: brick - modular components used for data visualization using WebUI-HTML, python (.py - Fig. 1) - the main algorithm based on AI, sketch (.ino - Fig. 1) with a program written in C used to operate the MCU and a yaml (serialization language) file. In the project directory structure in Python, there is a text file named "requirements.txt" containing a list of the necessary libraries required to implement the intelligent algorithms. The use of tools such as Google Colab allows for the full visualization and validation of the algorithm (test code) before its implementation on the MPU unit (base code). Early verification of the program (test code) allows for the avoidance of many errors and the optimization of performance. The trained model was saved using the joblib library, which enables fast loading in the SBC environment and its use for real-time inference on continuously incoming sensor data packets. This approach eliminates the need to retrain the model after each system boot and reduces the computational load on the device.

On the electronic devices market, computing systems containing advanced NPU (Neural Processing Unit) or TPU (Tensor Processing Unit) neural processing units are manufactured. A description of such systems is presented in [3]-[4]. Advanced digital systems have the capability to accelerate computations related to artificial intelligence and possess computational power starting from 0.5 TOPS (Tera Operations Per Second). Modern systems equipped with wireless communication allow for shifting computations to the cloud or for edge computing, especially in IoT/IIoT (Internet of Things / Industrial Internet of Things) systems. A problem in such systems is ensuring appropriate security of the transmitted and processed data. Systems containing a GPU (Graphics Processing Unit) are very fast for AI computations, but due to costs and electrical power consumption, they are not considered by the author of this article. The main advantage of an AI application accelerator compared to a standard SBC (based on ARM Cortex-A (Application processors) and Cortex-M (Microcontrollers) processors) is the support for matrix operations, which results in the efficient execution of algorithms. Description of the Arm Cortex-A53 system is in [5].

A. Literature review

The article [6] presents an analysis of popular single-board platforms (SBCs and microcontrollers) and their integration with various sensor technologies. A systematic review of the applications of machine learning and deep learning is presented in [7]. This research was used to support the selection of the SBC platform in the present work. A comparative study on SBCs for AI and deep learning (DL) implementations can be found in [8]. The selection of hardware for DL became an important aspect. An experimental comparison between SBCs and FPGAs is described in [9]. The study is important due to the possibility of significantly speeding up computations by using an FPGA with an accelerator with a small increase in power consumption. The study in [10] provided a comprehensive

performance analysis of SBC technology as IoT server. Communication within an IoT system is also an important aspect, as it opens new possibilities for extending and enhancing the proposed architecture. A design and evaluation of an SBC-based power monitoring system was presented in [11]. The proposed system uses a power monitoring sensor that measures voltage and current (using a resistive shunt). The measured signals can also be used to monitor the operating status of devices in the system proposed in the present work. The book [12] shows how to use SBC technology. The article shows the real application of SBCs in measurement systems [13]. SBCs are capable of performing tasks that previously required significantly more expensive solutions such as a PC (Personal Computer). The implementation of a data repository on a Raspberry Pi was given in [14]. This paper demonstrates the creation of an advanced and reliable time-series data repository on an SBC, ensuring its persistent storage. This illustrates how sophisticated a system can be implemented. Real-time deep learning model deployment on Raspberry Pi was given in [15]. The information contained in this article allows for practical implementation of an AI model (lightweight deep learning model) on SBC, with model quality assessment using MAE, MAPE and RMSE metrics. Model quantization was presented in [16]. Quantization effectively minimizes inference latency and energy use without decreasing accuracy, which is essential for resource-constrained platforms. Implementation of artificial neural network on Raspberry Pi was given in [17]. This work presents a practical implementation of an Artificial Neural Network (ANN) on the Raspberry Pi platform, demonstrating that this microcomputer can process signals in real time. The evaluation methodology presented in [18], including the use of metrics such as recall, precision, accuracy and F1-score, provided guidance for assessing the performance of the proposed model in the present study. The article [19] provides an overview of machine learning algorithms, including artificial neural networks and decision trees, which were considered during the development of the proposed solution. The description of decision trees is given in [20]. This article was important for the present work, as it provided essential information on the CART and C4.5 algorithms. The concepts of decision region and decision boundary are presented in this paper [21]. This article was important in the present work as it defines and explains how decision boundaries are formed models.

II. OBJECTIVES

The article attempts to apply neural networks in SBC (Single Board Computer) systems running under the Linux operating system. The main objective of the article is to develop the field of intelligent algorithms deployed in digital systems. The application of appropriate data processing systems enables the implementation of complex artificial intelligence models used for data classification. In particular, this study aims to find answers to the following questions:

1. How does using the Google Colab cloud environment at the design stage help validate the algorithm and support model selection before implementation on a resource-constrained hardware system?

2. What are the key methods for minimizing model complexity that affect response time and RAM utilization in the target hardware system?

The study of diverse algorithms and the use of tools supporting their creation allowed for expanding the technical and creative possibilities of implementing AI solutions in embedded systems.

III. MATERIALS AND METHODS

The measurement module monitoring the parameters of the tested device is the Nano 33 BLE Sense board. This system monitors operating temperature, vibration levels, and light intensity in real time, and analyzes signals from a digital microphone. The diagnostics are complemented by precise current measurement, enabling the AI algorithms to more accurately assess the technical condition of the monitored device. The data are transmitted to the MPU unit using an RPC (Remote Procedure Call) mechanism, which acts as an efficient communication bridge. An architecture designed in this manner enables the separation of the real-time data acquisition layer from the analytical Linux layer, where the implemented AI algorithms classify the operational state of the device based on sensor data. Fig. 3 shows the waveform of the load current I_{load} flowing through the terminals of the tested device in the 'NORMAL' operational state.

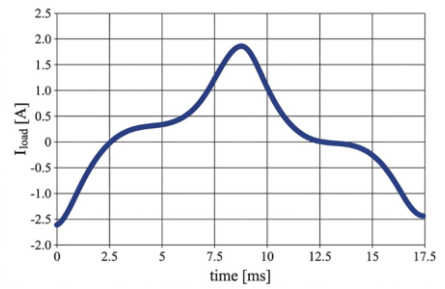


Fig. 3. The measured current I_{load} in the circuit of the tested device in the 'NORMAL' operational state.

The current distribution in the frequency domain is shown in Fig. 4, which was obtained for different sampling parameters and time window functions.

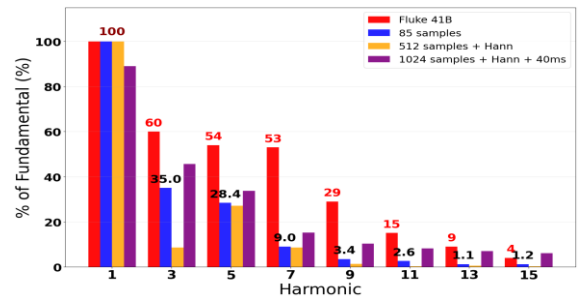


Fig. 4. FFT spectrum of the current waveform from Fig. 3 (Note: On the horizontal axis, the digit '1' represents the 50Hz fundamental component, used as the 100% reference point).

On the FFT graph (Fig. 4), the reference values obtained by using a Fluke 41B power quality analyzer are marked in red. Bars in the FFT distribution: blue, yellow, purple (Fig. 4) were obtained by using different digital methods (different number of samples and time windows – Hann (Hanning)). This analysis aimed to select the optimal parameters for the SBC system. It was observed that too small a number of samples (blue bar) or an inappropriate window (yellow bar) lead to the spectral leakage phenomenon and underestimation of the results, particularly for the 3rd and 7th harmonics. Ultimately, a configuration based on 85 samples was chosen for the extraction of the IHD parameter (individual harmonic distortion). Despite its lower resolution compared to the reference, it allows for the fastest determination of diagnostic features while maintaining sufficient precision, which is crucial for minimizing latency in the AI inference process within the MPU unit. The IHD_n factor is defined by the relationship $IHD_n = I_n/I_1$. For $n=3$, it is defined as the ratio of the RMS value of the third harmonic I_3 to the fundamental component I_1 . An appropriately selected number of samples and window functions allowed for the estimation of the IHD₃ parameter in a manner optimal for the system's computational resources, providing suitable input data for the neural network.

Example current waveforms of the tested device operating in different states are shown in the graphs below (Fig. 5).

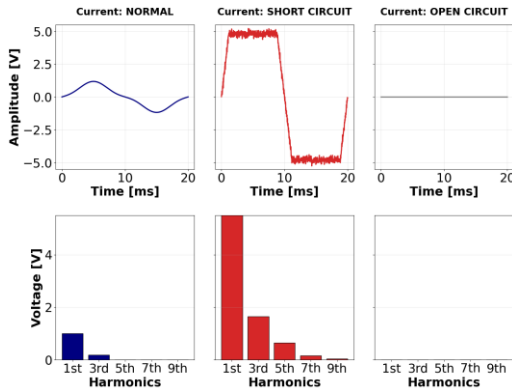


Fig. 5. Current waveforms under different operating states and their FFT spectrum distribution (respectively, the IHD3 coefficients are (18.08% (NORMAL), 27.5% (SHORT CIRCUIT), 0% (OPEN CIRCUIT))).

Fig. 5 shows three operating states of the device: under nominal conditions – normal, faulty (in this case a short circuit), and open circuit, as well as the corresponding FFT spectrum distributions. The second work state is represented by a short circuit, accompanied by rapid physical phenomena that can be detected using many parameters by the sensors of the Nano 33 BLE Sense module. In this case, besides the sudden increase in current amplitude and the distinct change in the FFT spectrum (IHD3 rising to 38%), intensified vibrations, discharge flashes at the terminals, and a rapid increase in component temperature are also observed. The acquired signals are preprocessed and stored in an array. The data are then transmitted to the MPU unit, where they are fed into the AI classification model.

For the diagnostics of the tested device with Arduino Nano 33 BLE Sense Rev2, transducers were used:

- temperature – (HTS221) used to determine the occurrence of excessive component temperature and can indicate a circuit overload (Fig. 6 – waveform: red continuous line

“Temp”). The output signal is transmitted via an I²C-type communication interface;

- digital microphone – (MP34DT05) – registers sound indicating operational fluency, wear and tear of mechanical components, or a system failure (Fig. 6 – waveform: orange continuous line with triangular markers “Mic”). The system provides a digital signal output in the PDM (Pulse-Density Modulation) format;
- Vibrations (BMI270) -an accelerometer is used to determine the device’s dynamic mechanical state and structural stability (Fig. 6 – waveform: blue continuous line “Vibrations”). The output signal is transmitted via an I²C interface;
- Ambient light intensity detection (APDS9960) – used to register an increase in light intensity (in lux), which provides information to the AI algorithm regarding the occurrence of a discharge (Fig. 6 – waveform: green dashed line “Lighting”). The output signal is transmitted via an I²C interface.

All values from the measurement sensors listed above are given to the AI system, where they are analyzed. Sample time waveforms from the listed sensors are shown in Fig. 6.

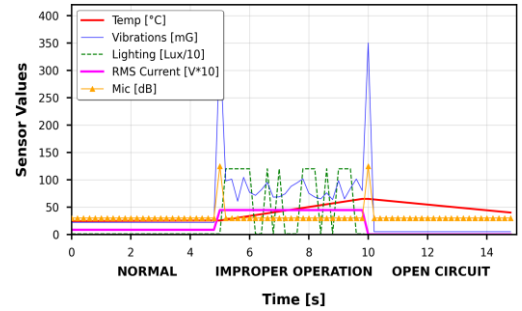


Fig. 6. Time waveforms of vibration, temperature, and lighting, as well as the root mean square (RMS) current values, obtained from the tested device with detected operating state.

The current value obtained from the output of the current transducer is converted to the root-mean-square (RMS) value and is shown in Fig. 6. The labels under the graph – formatted as text strings: NORMAL, IMPROPER OPERATION, or OPEN CIRCUIT – represent the algorithm's response to the provided data.

Use of the data from multiple sensors (temperature, vibrations, sound) is essential because it allows for the detection of abnormal operating conditions that predict the onset of a failure, such as a short circuit or an open circuit, even before they actually occur.

A. Algorithms implemented in the SBC system

1) Decision Tree

A decision tree is a graphical method for supporting the decision-making process. It is one of the most frequently used data analysis techniques. The tree consists of a root, nodes, and branches. The classification process using decision trees is computationally efficient. A tree to determine the category of a case requires testing only selected attributes along the decision path. Based on the input data originating from the Arduino Nano 33 BLE Sense Rev2 and the current transformer [RMS current signal value from the current transformer, temperature, vibrations [mG], light intensity value [lux], value from the digital microphone [dB]], the decision tree shown in Fig. 7 was

generated. Tools like Google Colaboratory allow running code, which enables precise testing and visualization of the algorithm's operation. The graph in Fig. 8 shows the dependence of the AUC metric (Area Under the ROC Curve – a measure of classification quality) on the max_depth parameter (maximum depth) for the decision tree model. For a depth of 1, the model achieves an AUC of approximately 0.87. Starting from a depth of 2 and for all higher values (3–8), the model achieves maximum, ideal efficiency with an AUC = 1.0.

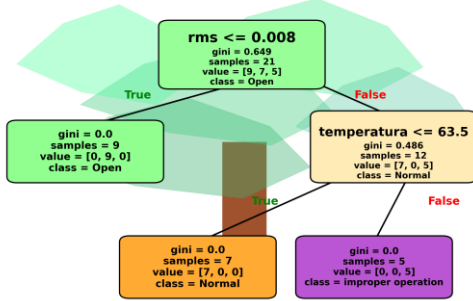


Fig. 7. Diagram of the decision tree structure. (The diagram was generated in Google Colab).

Fig. 8 shows how the classification quality changes with an increase in the maximum depth of the tree. At depth 1, the model still cannot properly separate the classes, which is why all metrics — AUC, F1, and Accuracy — have lower values. After increasing the depth to 2, a sharp jump occurs: the tree begins to correctly recognize all cases and achieves ideal results. Further increasing the depth no longer affects the performance of the model, as all the necessary information has been captured at the level of two splits.

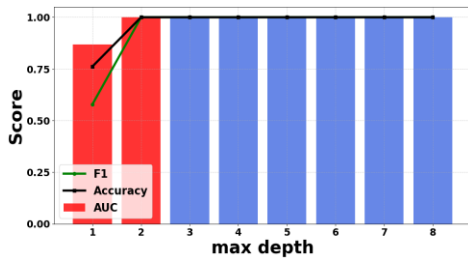


Fig. 8. Plot of classification quality (AUC) versus tree depth max depth – maximum depth of the decision tree. (The chart was generated in Google Colab).

Based on the presented tree structure (Fig. 7), it can be observed that the decision-making process relies on sequentially checking only two parameters: the RMS current value and temperature. These values effectively classify all 21 training samples into three separate categories without any errors. Further increasing the depth no longer affects performance. This visualization clearly confirms that the algorithm requires at most two logical comparison operations to uniquely classify the state of the device as safe normal (Normal), or critical (Improper operation).

In practice, this analysis enables the precise selection of the classifier's architecture before the implementation phase on hardware. Proving that a model with minimal complexity achieves 100% accuracy eliminates the need to deploy more resource-intensive algorithms on the SBC platform, ensuring optimal management of its memory and processor cycles. The plot shown in Fig. 9 illustrates how the classification model

separates the data into different classes (Class index: 0, 1, 2) based on two parameters: current ($V[V_{rms}]$, from the output terminal of electronically assisted current transformer) and temperature (Temperature $^{\circ}C$, from the HTS221 sensor). The parameters within the decision tree blocks (Fig. 7) — Gini impurity, samples, value, and predicted class — represent the metrics that describe how the decision tree makes its choice.

The plot in Fig. 9 displays the three-dimensional decision surface of the classification tree, demonstrating how the model divides the measurement space into three classes depending on the RMS and temperature values.

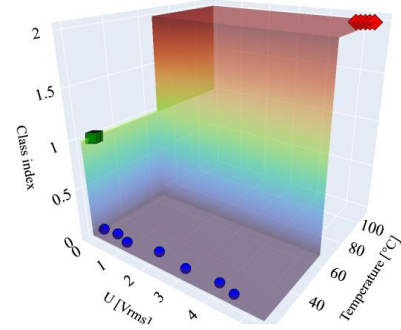


Fig. 9. Three-dimensional decision surface of the tree classifying the system state, where the Z-axis represents the class index: 0 = Normal (blue), 1 = Open (green), and 2 = Short/improper operation (red). (The three-dimensional decision surface was generated in Google Colab).

The colored surface illustrates the boundaries between classes, and the data points (green, blue, and red) show the actual samples assigned respectively to the classes: Normal (0), Open (1), and Short/improper operation (2). The use of the Google Colab environment made it possible to run simulations and generate graphs showing the dependence of metrics (F1-score, AUC, Accuracy) on the structural parameters of the model (maximum depth - max_depth) prior to its physical implementation on the SBC platform. This visualization enables a clear assessment of the algorithm's stability and operational correctness, which serves as a justification for the final, optimized configuration of the embedded system.

The implementation of the classifier, both in the test environment (Google Colab) and on the target hardware platform (SBC/MPU), was based on the DecisionTreeClassifier class from the scikit-learn library. The package installation process on the Linux system was executed using a requirements.txt configuration file (containing the necessary libraries). The version of the code intended for the base system—the AppLab environment—was optimized by removing the matplotlib library. Transferring the code did not entail any loss of functionality — the model retained its full predictive capability.

Answer to Question 1: The use of the Google Colab environment enabled the verification of the tree depth and the analysis of its impact on classification metrics. Based on this, it was demonstrated that restricting the model structure to just 2 levels yields maximum efficiency ($F1 = 1.0$). This allowed for the optimization of the algorithm well before its physical deployment on the SBC platform.

Answer to Question 2: The deployment of the decision tree model via ONNX serialization and INT8 quantization yielded

significant performance improvements. Transitioning from the standard scikit-learn format to ONNX reduced the serialized model size from 0.0017 MB to 0.0011 MB, optimizing memory utilization for embedded systems. Furthermore, eliminating the heavy library overhead accelerated the inference speed. The execution time per single sample was shortened from 528.32 μ s (scikit-learn) to just 62.56 μ s (Tree ONNX INT8). This represents an approximate 8.4-fold speedup, measured directly from the moment the data vector enters the model input session to the generation of the classification output. Despite this quantization and complexity reduction, the optimized model maintained 100% effectiveness.

2) Multilayer Perceptron (MLP) Neural Network

The multilayer perceptron (MLP) network in the proposed SBC system processes six input features and performs classification into three output classes (NORM, SHRT, OPEN). The proposed neural network consists of two hidden layers, which allows it to effectively model complex, nonlinear relationships between input parameters and the system's operating state labels.

The input of the MLP network is a vector containing the data [RMS, H3/H1=ratio, Temperature, Vibration (Vib), microphone (Mic), lighting (Lux)]. The dimensions of the MLP network architecture are $6 \times 10 \times 5 \times 3$ and include two hidden layers (Hidden_1 x Hidden_2: 10×5) with the structure shown in Fig. 10.

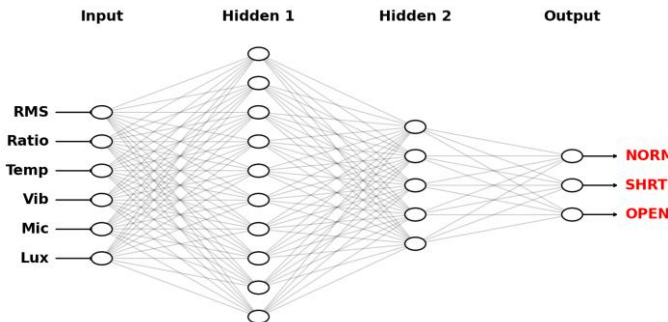


Fig. 10. Architecture of the MLP network for input signal classification.

To compare the efficiency of the algorithms, an MLP network with an identical architecture was implemented using both the scikit-learn and TensorFlow libraries. Despite the identical model structure, differences appear in the low-level implementation, weight updates, and optimizer configuration options, resulting in different characteristics of the learning process. Therefore, with the same network architecture, a TensorFlow model can achieve convergence faster and more stably than scikit-learn.

The plot in Fig. 11 illustrates three loss function curves for different configurations and training methods of the MLP network. All curves show how the model error changes over successive training epochs.

A comparison of the loss curves is important because all models train on the same dataset. This allows for the evaluation of the convergence speed, training stability, and final error level for different configurations and frameworks. The plot (Fig. 11) shows how different optimizers and implementations affect the training dynamics, allowing for the selection of the model best suited for further use.

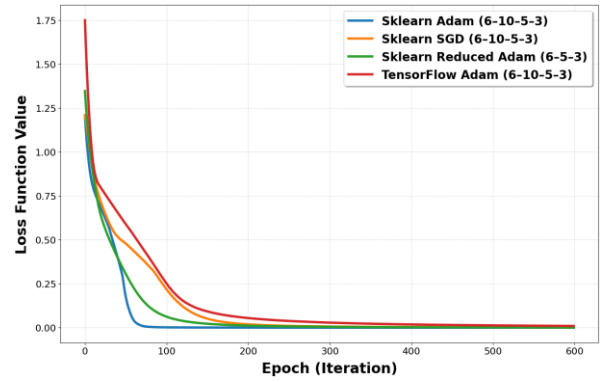


Fig. 11. Loss function for an MLP network (comparison of models training on the same data). (The chart was generated in Google Colab).

A. Neural network exported to ONNX format

ONNX (Open Neural Network Exchange) is an open exchange format for models that enables the implementation of neural networks across different frameworks – e.g., from PyTorch or TensorFlow to dedicated runtime environments on SBC platforms. Thanks to this, the model training process can be conducted in the efficient Google Colab cloud environment, and the complete model — including its architecture and weights — can then be exported into a single ONNX file. This format allows for the flexible transfer of neural network structures between different programming environments without the need to retrain the network.

The application of the ONNX format and the dedicated ONNX Runtime engine on the target SBC platform brings key advantages from the perspective of optimizing embedded system operations. First of all, this solution reduces RAM usage and decreases the computational load on the single-board computer's processor, because the lightweight execution engine only loads the mathematical structure of the network, completely eliminating the resource burden caused by heavy training libraries. Additionally, ONNX improves the reproducibility of classification results by avoiding discrepancies that may arise from using different software versions in the cloud and on the edge device.

Answer to Question 1: The Google Colab environment made it possible to conduct experiments on the MLP network in multiple configurations and to compare its implementations in scikit-learn and TensorFlow, which enabled effective model optimization prior to deployment on the SBC platform. This approach also allowed the verification of the algorithm's correctness and the selection of the most efficient architecture already at the design stage, without burdening the limited resources of the target device.

Answer to Question 2: The baseline scikit-learn MLP required 1389.40 μ s per single inference with an accuracy of 95.2%. In comparison, the TensorFlow/Keras implementation showed heavy library overhead, inflating the execution time to 12342.74 μ s and significantly reducing accuracy to 72.6%. Migrating the model to the ONNX Runtime engine successfully achieved a computational speedup, dropping the execution time to just 95.88 μ s. However, this format conversion triggered an accuracy drop to 92%. Furthermore, applying INT8 quantization to the ONNX model did not restore accuracy and actually increased the execution time to 113.85 μ s.

IV. RESULTS

All four neural network configurations achieved full convergence, and the loss function dropped to almost zero before the 600th epoch. The initial experimentation was conducted in Google Colab, which enabled rapid prototyping and model training. After porting the implementation to the target SBC system, the models retained full functionality, but their sizes varied significantly depending on the format. The TensorFlow representation occupied 26.6 KB, while ONNX (MLP models trained in scikit-learn, 6x10x5x3) reduced the size to 1.9 KB, and the use of INT8 quantization (reduced MLP model 6x5x3) reduced the file size to just 0.5 KB.

V. DISCUSSION

The obtained results confirm stable convergence of all models, which is consistent with prior knowledge regarding small MLP networks for simple classification tasks. Optimizations such as exporting to ONNX and INT8 quantization align with established model reduction strategies for embedded systems. The resulting file sizes demonstrate that even highly simplified architectures can be effectively implemented on an SBC with minimal memory consumption. Furthermore, the use of cloud environments such as Google Colab enabled rapid prototyping, while the actual performance testing was conducted directly on the target SBC.

VI. CONCLUSIONS

The experiments confirmed that the intended research goal was achieved, and the developed models met the requirements for convergence and computational efficiency. The results showed that the optimization techniques used allowed for a significant reduction in model size while maintaining their functionality on the target SBC. Ultimately, the proposed approach was confirmed to be effective and can be successfully implemented in resource-constrained embedded systems.

VII. SOFTWARE USED

The following software was used to complete the tasks in this article: Google Colab with Python for algorithm development, Arduino App Lab with Python and C for testing on the SBC platform, and the Debian Linux operating system running on the SBC device. Figures 3 through 11 were generated in the Google Colab environment.

REFERENCES

- [1] Hsiao, Y.-S., Bhattacharya, S., & Halder, A. "Arduino Uno Q with Qualcomm AI Chip: Enabling Next-Generation Edge Intelligence for Embedded AI Prototyping". *Asian Journal of Electrical Sciences*, 14(2), 6–20, 2025. <https://doi.org/10.70112/ajes-2025.14.2.4279>
 - [2] P. B. Hemanthkumar, S. R. Anireddy, F. T. Josh, and R. Venkatesan, "Introduction to ARM Processors & Its Types and Overview to Cortex M Series with Deep Explanation of Each of the Processors in This Family" in 2022 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2022. <https://doi.org/10.1109/ICCCI54379.2022.9740768>
 - [3] Zhao, HaoMiao. "Optimized Design of Lightweight NPU Accelerator for the Internet of Things Based on Mixed-precision convolution and Systolic Array". 2026. <https://doi.org/10.21203/rs.3.rs-8705428/v1>
 - [4] P. Lin, H. Zhang, L. Li, and Y. Li, "Systolic Array Architecture for Multitasking Neural Processing Unit," in the 2024 29th International Conference on Automation and Computing (ICAC), Sunderland, UK, 2024. <https://doi.org/10.1109/ICAC61394.2024.10718735>
 - [5] B. Schwaller, B. Ramesh, and A. D. George, "Investigating TI KeyStone II and Quad-Core ARM Cortex-A53 Architectures for On-Board Space Processing," in 2017 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2017. <https://doi.org/10.1109/HPEC.2017.8091094>
 - [6] Álvarez, J. L., Mozo, J. D., & Durán, E. "Analysis of Single Board Architectures Integrating Sensors Technologies". *Sensors*, 21(18), 6303. 2021. <https://doi.org/10.3390/s21186303>
 - [7] Aldahoud, A., Fezari, M., Al-Dahoud, A., & Aqel, D. "A Systematic Review on Applying Machine Learning and Deep Learning on SBCs". *WSEAS Transactions on Information Science and Applications*, 21, 272–281. 2024. <https://doi.org/10.37394/23209.2024.21.26>
 - [8] Al-Dahoud, A., Fezari, M., & Aldahoud, A. "Comparative Study on SBC for AI and DL implementation". 2023. <https://doi.org/10.13140/RG.2.2.20232.19205>
 - [9] Tamburello, M., Caruso, G., Adami, D., & Giordano, S. "Experimental Comparison Between SBC and FPGA for Embedded Neural Network Acceleration". *ICC 2023 - IEEE International Conference on Communications*. 2023. <https://doi.org/10.1109/ICC45041.2023.10279670>
 - [10] Rohadi, E., Rahmad, C., Sukmana, S. E., Sartimbul, A., Ping, K. A. H., Joseph, A. A., Mat, D. A. A., Rahmat, M. F., Zakki, A. A., & Ramadhan, D. N. "A review: Performance Analysis of Single Board Computer Technology as an IoT System". 2025 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT), 106–110. 2025. <https://doi.org/10.1109/IAICT65714.2025.11101682>
 - [11] Guegan, L., Tofaily, S., & Raïs, I. "Design and Evaluation of Single-Board Computer Based Power Monitoring for IoT and Edge Systems". 2023 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), 716–723. 2023. <https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics60724.2023.00127>
 - [12] Kanagachidambaresan G. R., "Role of Single Board Computers (SBCs) in Rapid IoT Prototyping," 2021. Springer. <https://doi.org/10.1007/978-3-030-72957-8>
 - [13] Leccese, F., Cagnetti, M., Di Pasquale, S., Giarnetti, S., & Caciotta, M. "A New Power Quality Instrument Based on Raspberry-Pi". *Electronics*, 5(4), 64. 2016. <https://doi.org/10.3390/electronics5040064>
 - [14] Samourkasidis, A., & Athanasiadis, I. N. "A Miniature Data Repository on a Raspberry Pi". *Electronics*, 6(1), 1. 2017. <https://doi.org/10.3390/electronics6010001>
 - [15] Bentaieb, S., Yassine Hamzi, K. E., Benmarouf, M. N., & Nehari, D. "Real-Time Deep Learning Model Deployment on Raspberry Pi Using Streamlit for Electricity Consumption Forecasting". 2024 IEEE International Multi-Conference on Smart Systems & Green Process (IMC-SSGP). 2024. <https://doi.org/10.1109/IMC-SSGP63352.2024.10919804>
 - [16] Xiao, P., Zhang, C., Guo, Q., Xiao, X., & Wang, H. "Neural networks integer computation: Quantizing Convolutional Neural Networks of Inference and Training for Object Detection in Embedded Systems". *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 17, 15862–15884. 2024. <https://doi.org/10.1109/JSTARS.2024.3452321>
 - [17] Bharadwaja, V., Ananmy, R., Nikhil, S., Vineetha, K. V., Shah, J., & Kurup, D. G. "Implementation of Artificial Neural Network on Raspberry Pi for Signal Processing Applications". 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 1488–1491. 2018. <https://doi.org/10.1109/ICACCI.2018.8554887>
 - [18] Xu, Z., Zhuang, X., & Ma, S. "Machine Learning-Assisted Systematic Review: A Case Study in Learning Analytics". *Education Sciences*, 15(11), 1488. 2025. <https://doi.org/10.3390/educsci15111488>
 - [19] Praba, R., Darshan, G., Roshanraj, K. T., & Surya Prakash, B. "Study On Machine Learning Algorithms". *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 67–72. 2021. <https://doi.org/10.32628/CSEIT2173105>
 - [20] Mienye, I. D., & Jere, N. "A Survey of Decision Trees: Concepts, Algorithms, and Applications". *IEEE Access*, 12, 86716–86727. 2024. <https://doi.org/10.1109/ACCESS.2024.3416838>
- Z. Yan and C. Xu, "Studies on Classification Models Using Decision Boundaries," 2009 8th IEEE International Conference on Cognitive Informatics, Hong Kong, China, 2009, pp. 287–294. <https://doi.org/10.1109/COGINF.2009.5250724>