On-device training of artificial intelligence models on microcontrollers

Bao-Toan Thai[1], Vy-Khang Tran[1], Hai Pham[1,2], Chi-Ngon Nguyen[1], Van-Khanh Nguyen[1]
[1]Faculty of Automation Technology, College of Engineering, Can Tho University, Can Tho, Vietnam
[2]School of Engineering, RMIT University, Melbourne, Australia

Article Info

Article history:

Received Nov 28, 2023
Revised Jan 26, 2024
Accepted Feb 14, 2024

Keywords:

Artificial intelligence
Free real-time operating system
Micro-controllers
On-device training
Real-time operating system

ABSTRACT

Numerous studies are currently training artificial intelligence (AI) models on tiny devices constrained by computing power and memory limitations by implementing model optimization algorithms. The question arises whether implementing traditional AI models directly on small devices like micro-controller units (MCUs) is feasible. In this study, a library has been developed to train and predict the artificial neural network (ANN) model on common MCUs. The evaluation results on the regression problem indicate that, despite the extensive training time, when combined with multitasking programming on multi-core MCUs, the training does not adversely affect the system's execution. This research contributes an additional solution that enables the direct construction of ANN models on MCU systems with limited resources.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Van-Khanh Nguyen
Faculty of Automation Technology, College of Engineering, Can Tho University
Campus II, 3/2 street, Ninh Kieu district, Can Tho city, Vietnam
Email: vankhanh@ctu.edu.vn

1. INTRODUCTION

The tiny machine learning (tinyML) model is an approach that enables the direct application of machine learning (ML) on embedded systems. It focuses on integrating ML techniques directly into micro-controller units (MCUs) with limited computing power and memory resources [1]. The tinyML approach has been applied in various fields such as healthcare, smart agriculture, environmental, and anomaly detection [2]–[4]. Most of these applications utilize models pre-trained on powerful computers and then deploy them directly onto MCUs. This provides flexibility and efficiency in processing information now on the device [5]. Based on the research in [6] the typical tinyML deployment process involves the following steps: training the model on a powerful computing device, quantizing the model within the TensorFlow Lite framework [7], and finally deploying the quantized model on an MCU to perform inference tasks.

However, when needed, the tinyML model cannot be retrained with new data. Currently, some artificial intelligence (AI) models in general, ML and deep learning (DL) models in particular have been directly trained on MCUs, garnering significant interest [8], [9]. This approach is also referred to as on-device training (ODT). ODT directly trains the model on small computing devices, such as MCUs, without pre-training. This method lets the model be instructed with data acquired during the device's operation [9]. However, ODT still faces challenges like computational capability and memory constraints. Many recent studies, such as [10]–[14] have proposed various techniques to optimize models and memory, enabling the computation of complex models on small devices.

Real-time operating system (RTOS) is an operating system that supports scheduling mechanisms to ensure tasks can be completed within specific time constraints [15], [16]. FreeRTOS [17] is one of the RTOS

kernels developed for embedded systems, and it supports the most common micro-controller families. Using freeRTOS not only helps manage tasks more efficiently but also leverages the capabilities of multi-core MCUs for parallel execution. This enhances the performance of embedded devices. FreeRTOS has been applied to accelerate the computation of artificial neural networks (ANN) by dividing them into two corresponding tasks and assigning them to two cores for scheduling [18]. It is also applied to enhance the efficiency of signal preprocessing for tinyML applications [19]. Accelerating tinyML with freeRTOS is challenging due to the undisclosed structure of pre-trained models. However, for ODT, freeRTOS can effectively leverage its capabilities when running on multi-core MCUs, scheduling training and prediction tasks on two separate cores. This allows devices to make continuous predictions without interruption from the model training process when needed.

Current ODT solutions still have some inherent limitations. According to Sudharsan *et al.* [20], the Train++ algorithm is utilized to address classification and regression problems on common MCUs. However, the embedded program algorithm needs to be more specific and generalized to accommodate the addition of hidden layers. As a result, creating complex ANNs is challenging. Craighero *et al.* [21] has developed the capability to train convolutional neural network (CNN) models directly on the STM32 MCU to address human action recognition problems. However, deploying this research might face challenges if the data cannot be classified, for example, in cases like electrocardiogram (ECG) signals requiring cardiac expert evaluation. This study also focuses on a specific application and has only been tested on one MCU. This suggests that if training ANN models using unsupervised learning algorithms on MCUs is possible, the applicability could be broader, such as models for anomaly detection based on autoencoders, and ANN models for prediction in internet of things (IoT) applications.

This research focuses on developing a feature-rich and highly customizable capability for creating and training ANNs on embedded systems. To achieve this, fundamental functions of ANN models such as forward and backward propagation, activation functions, loss functions, and ANN creation and training functions (i.e., add, use_loss, fit and predict) will be generically programmed based on object-oriented programming languages. Subsequently, these functions will train and predict on one PC and some common MCUs to evaluate performance. Additionally, freeRTOS will be applied to perform parallel training and prediction tasks on multi-core MCUs to enhance the efficiency of the ODT method.

2. EXECUTION METHODS

First, Figure 1 illustrates the relevant mathematical analyses of the neural network model. Subsequently, the ANN library is created based on them. Next, several ANN models ranging from simple to complex are directly implemented on both one PC and various types of MCUs using this library to evaluate their performances.

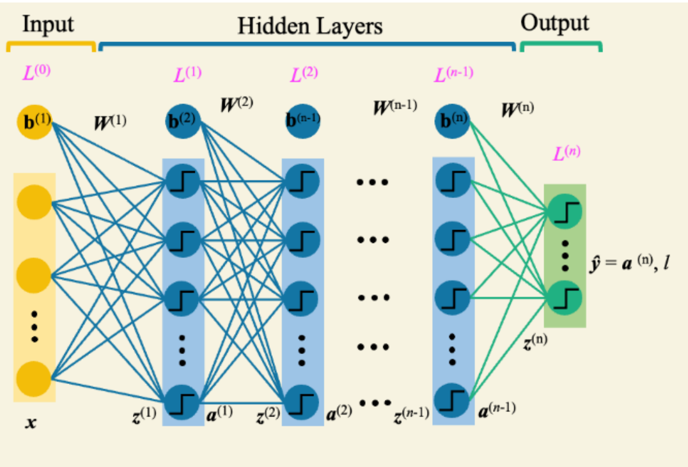

Figure 1. Overal neural networks

2.1. Related mathematics

An overview of the ANN [22] model is illustrated in Figure 1. In an ANN model, we typically have an input layer and an output layer, with multiple hidden layers in between, depending on the choice to suit

the characteristics of the data. To provide a more precise understanding, Figure 2 illustrates each layer's forward propagation computation process. Notably, each layer's output becomes the following layer's input, forming a linked chain between layers in the model. The relevant mathematical symbols of the AI model are depicted in Figure 1. x represents the input data, W is the weight matrix, b is the bias parameter, z is the linear parameter, a is the activation function, $\hat{y}$ is the output of the model, l is the loss function, L represents the number of layers in the model, and n denotes the number of layers in the model.

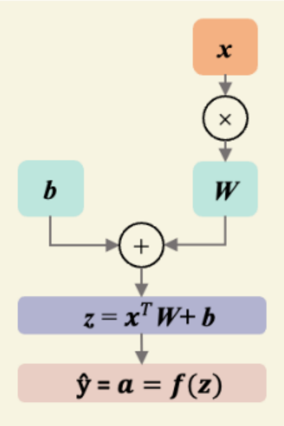

Figure 2. Forward propagation for each layer

The output of each layer is calculated according to (1):

$$a = f(z) \quad (1)$$

where a is the activation function, which includes non-linear functions such as Relu, Tanh, and Sigmoid [23]. z being the value of the linear function calculated by (2):

$$z = x^T W + b \quad (2)$$

During the model training process, the backward propagation algorithm is applied. This algorithm starts from the last layer to the first layer, combining the chain rule. In this process, the gradient of the loss function is calculated in (3) to adapt the parameters W, x and b to the data, as presented in [21].

$$\frac{\partial l}{\partial W} = \frac{\partial l}{\partial \hat{y}} x^T, \quad \frac{\partial l}{\partial x} = \frac{\partial l}{\partial \hat{y}} W^T \odot f'(x), \quad \frac{\partial l}{\partial b} = \frac{\partial l}{\partial \hat{y}} \quad (3)$$

The model parameters are updated by the stochastic gradient descent (SGD) [24] algorithm, as presented in (4).

$$\theta = \theta - lr \frac{\partial l}{\partial \theta} \quad (4)$$

where θ is the set of parameters to be optimized, such as W, b, x, and lr is the learning rate.

After each training cycle, the loss function is used to evaluate the training performance. In this study, three types of loss functions have been implemented, including mean squared error (MSE) [25], binary cross-entropy (BCE), and categorical cross-entropy (CE) [26], represented mathematically in (5) to (7).

$$MSE(y, \hat{y}) = \frac{1}{M}(y - \hat{y})^2 \quad (5)$$

$$BCE(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \quad (6)$$

$$CE(y, \hat{y}) = -y \log(\hat{y}) \quad (7)$$

where $\boldsymbol{y}$ is the actual value, $\hat{y}$ is the predicted value, and M is the total number of samples.

2.2. Programming artificial neural network library on micro-controller units

According to the mathematical analysis of the ANN model, the library is built with three classes and four methods, as presented in Table 1. The FClayers class performs computations for both forward and backward processes. The ActivationLayer class integrates non-linear activation functions. Finally, the Network class is the main class of the ANN library, comprising four main methods: add is used to add layers and activation functions to the model; use_loss is used to define the loss function for evaluating the model's performance; fit and predict are methods for training and predicting the model, respectively. The add method can dynamically define the number of neurons, making the ANN model creation highly flexible.

Table 1. Classes and methods of the ANN model

Class/Method	Function	Description
FCLayer	Class	As a fully connected layer in the neural network.
ActivationLayer	Class	The activation functions used in the network include: ReLU, Tanh.
Network	Class	The main class for managing the neural network.
add	Method of Network	Add a new layer to the neural network model.
use_loss	Method of Network	Specify the loss function for the model, including MSE, BCE, CE.
fit	Method of Network	Train the neural network model.
predict	Method of Network	Predict the output of the model.

An descriptive understanding of the ANN model implementation method on an embedded device is shown in Figure 3. Figure 3(a) illustrates a code snippet to create an ANN model with one input neuron, one output neuron, and two hidden layers. Firstly, the model object of the ANN model is instantiated using the construction method of the Network class. Next, the add method is used to create the input neurons for the two hidden layers (one layer with 64 neurons and the other with 32 neurons). Following each command to create a hidden layer is a command to add an activation function for the neurons in that layer, as demonstrated by adding the ReLU activation function. Finally, the command to create the output neuron and declare the loss function using the use_loss method. The ANN model will be trained by the fit method with the training data x_train, y_train, learning rate lr, and epochs number of iterations. The predict method is used to make predictions with new data. To visualize Figure 3(b) depicts the structure of an ANN model that has been implemented in Figure 3(a).

```
float lr = 0.001;
int epochs = 30;
Network model;
model.add(new FCLayer(1, 64));
model.add(new ActivationLayer(relu_activation, relu_prime));
// Can add additional hidden layers as needed.
model.add(new FCLayer(64, 32));
model.add(new ActivationLayer(relu_activation, relu_prime));
model.add(new FCLayer(32, 1));

model.use(mse, mse_prime);
model.fit(x_train, y_train, epochs, lr);
model.predict(data_new);;
```

(a)

Input Hidden Layers Output

R^1 R^{64} R^{32} R^1

(b)

Figure 3. Implement an ANN model on an embedded system: (a) program segments to train and predict the ANN model and (b) ANN model structure to create

The algorithm for the fit method is presented in Algorithm 1, executed in four steps. Firstly, err is initialized to 0. Subsequently, each sample of the x_train data is utilized to compute ForwardPropagation to find the output of the ANN with the current set of weights and biases. Next, the error is calculated relative to the actual values using the defined loss function. Simultaneously, the rate of change of the loss, derivative_loss, is computed. Finally, the BackwardPropagation algorithm is executed based on derivative_loss (i.e., $\frac{\partial l}{\partial \theta}$ in (4)), and the learning rate (lr) is used to update the parameters of the network. The optimization algorithm SGD is explicitly implemented as follows:

weights[*i*] -= *lr* * *weights_error*[*i*][*j*];
bias[*i*] -= *lr* * *output_error*[*i*];

In the equations above, weights, bias, weights_error, output_error correspond to W, b, $\frac{\partial l}{\partial W}$, $\frac{\partial l}{\partial b}$; where i, j range from 1 to n. The training algorithm is iterated for epochs times. The value of err after each epoch is collected to assess the success of the training process.

Algorithm 1. Training
```
fit (x_train, y_train, epochs, lr):
   For e ← each value of epochs:
     err ← 0
     For i ← each row of x_train:
       output[i] ← ForwardPropagation(x_train[i])
       err ← err + loss(y_train[i], output)
       BackwardPropagation (derivative_loss, lr)
     End For
     err ← err / number of samples
   End For
```

The predict method is presented in Algorithm 2. This method calls the ForwardPropagation method as in (2) to find the output of the trained ANN. This method evaluates the ANN model after training with a test dataset or new data collected by devices applying the ANN model created by this library.

Algorithm 2. Predict
```
predict (data):
   For i ← each row of data:
     output[i] ← ForwardPropagation(data[i])
   End For
```

2.3. Deployed on micro-controller units

To deploy the ANN model on the selected MCUs presented in Table 2, the Arduino integrated development environment (Arduino IDE) uploads the code directly to the MCUs. This ensures the flexible transferability of the model from a personal computer to the MCUs. Most chosen MCUs come from manufacturers with diverse CPU architectures, frequencies, and memory characteristics. This can assess the compatibility of the model with various hardware conditions.

Table 2. MCUs are selected for implementation

Name	Development Board	CPU	Frequency (MHz)	Flash (MB)	RAM (KB)
MCU-1	ESP32-Wroom32	Dual core Xtensa®32-bitLX6	240	4	520
MCU-2	Arduino nano 33 BLE	Single core Arm Cortex-M4F	64	1	256
MCU-3	Sipeed Maix Bit K210	Dual core Kendryte K210	400	16	8192
MCU-4	Raspberry Pi Pico RP2040	Dual core Arm Cortex-M0	133	2	264
PC	Macbook M1 2020	Apple M1	3.2(GHz)	-	8GB

The three proposed models to be evaluated on both MCUs and one PC are summarized in Table 3. All these models have one neuron in the input and output layers. Models 1, 2, and 3 have 1, 2, and 3 hidden layers, respectively, with corresponding parameter counts of 193, 4,353, and 8,513 parameters. These models will be implemented to evaluate metrics such as training and prediction time for a single input sample. Each model is trained for 30 epochs with a learning rate 0.001, using ReLu as the activation function and MSE as the loss function.

Table 3. MCUs are selected for implementation

Name	Construct	Parameter
Model-1	1×64×1	193
Model-2	1×64×64×1	4,353
Model-3	1×64×64×64×1	8,513

The performance of the ANN models is evaluated using a regression problem with a cubic polynomial of the form presented in (4). The training data, randomly generated from this polynomial, is

illustrated in Figure 4. The black dots on the graph represent the data used during training, and the red line represents the actual values of the polynomial.

$$g(x) = ax^3 + bx^2 + cx + d \quad (8)$$

The function $g(x)$ is defined as a cubic equation with coefficients a, b, c, d corresponding to 1, 2, -3, -4, respectively.

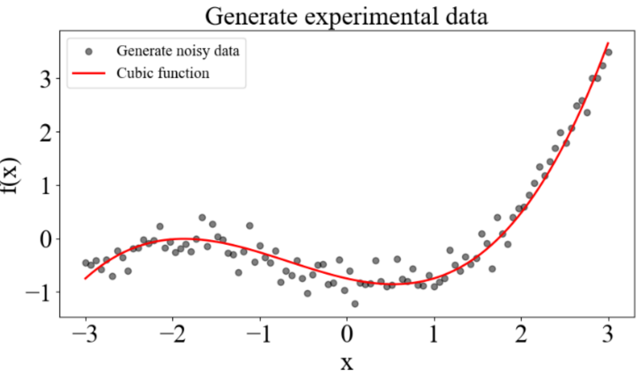

Figure 4. Data used to evaluate the model

Two datasets were created to evaluate the performance and efficiency of multi-threaded programming in the training and prediction of ANN models. The specific information about the two datasets is as follows. Dataset-1 contains 150 data points (x, $g(x)$), with x ranging from -3 to 3, divided into two sets: train and test, consisting of 100 and 50 data points, respectively. The MSE of the model is used for evaluation on both PC and MCU. Dataset-2 consists of 300 data points (x, $g(x)$), with x ranging from -3 to 3, divided into two sets: train and test, consisting of 200 and 100 data points, respectively. Particularly for Datas et-2, it is further divided into four smaller datasets, each containing 50 samples. These smaller datasets are sequentially used to train the model through Task Train, evaluating the parallel execution capability during training and prediction on a multi-core MCU.

The models will be trained and tested with single-task programming running on a single-core MCU or a separate core of a multi-core MCU and multi-task programming using freeRTOS to run on a multi-core MCU. Figure 5 presents two flowcharts implementing the ANN model on single-core MCUs and multi-core MCUs. Figure 5(a) illustrates the process of deploying the ANN model using sequential programming. The program will create the ANN model upon startup using the provided functions. If this is the first training, weights, and biases will be randomly initialized, and the model will be trained with Dataset-1. Conversely, the weights are loaded from the system's non-volatile memory if the ANN model has already been taught. Then, the program will perform the main loop, consisting of two sequential tasks: retraining the ANN and making predictions. These tasks are executed at different speeds, set by prediction and training time. In this case, it is evident that when the MCU retrains the system, it cannot predict new data.

To address the challenges, Figure 5(b) illustrates the process of training and predicting algorithms on a multi-core MCU using multi-task programming with freeRTOS. The training and prediction processes are implemented as two freeRTOS tasks, each assigned to a separate core of the MCU. FreeRTOS will be responsible for scheduling these two tasks to run in parallel, ensuring that the training process does not interrupt the prediction process. Similar to the algorithm in Figure 5(a), the program will create the ANN model using the provided functions after startup. If it is the first training, weights and biases will be randomly initialized, and the flag wait is set to true to notify that the prediction process must wait until the first training process is completed. Conversely, suppose the model has already been trained. In that case, the model's parameters will be loaded from the MCU's non-volatile memory, and the flag wait will be set to false to allow the task predict to operate. After each completion of the Task Train, the weights and biases will be saved to the MCU's non-volatile memory; the trained flag will be set to true to allow the task predict to update the new parameters, and then reset the trained flag to false.

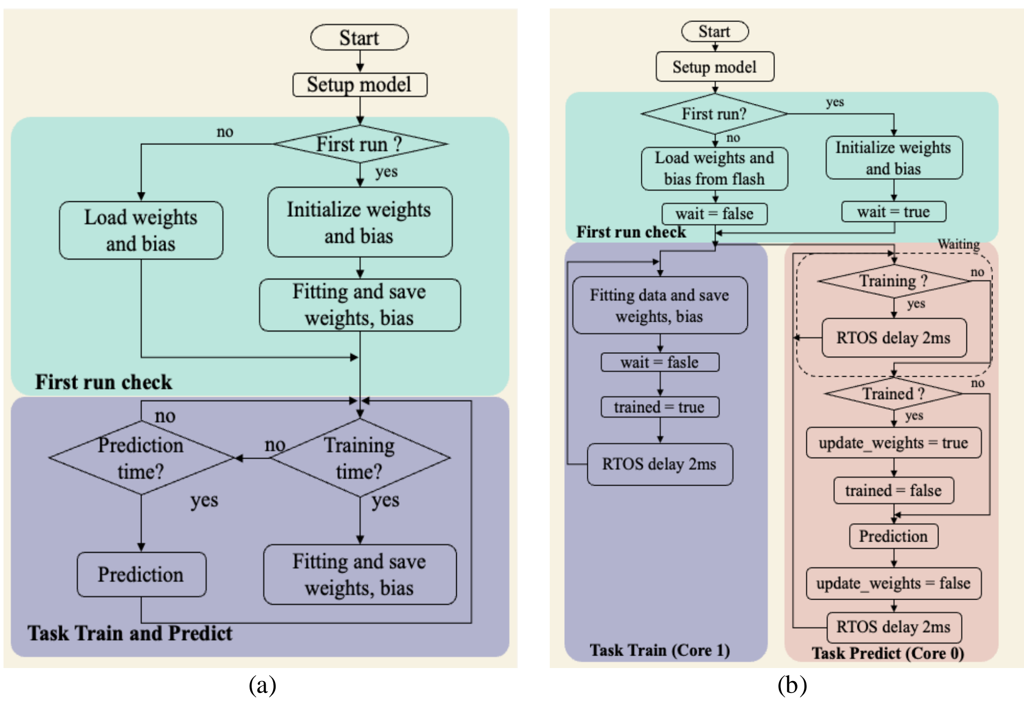

Figure 5. Flow chart of implementing ANN model: (a) single-core MCU, and (b) dual-core MCU

3. RESULTS

3.1. Evaluate artificial neural network performance on micro-controller units

The real-time execution time evaluation results for the three models on Dataset-1 are presented in Tables 4 and 5. In Table 4, MCU-1 and MCU-3 exhibit similar training speeds, significantly faster than MCU-2 and MCU-4. This difference could be attributed to the lower clock speeds of MCU-2 and MCU-4 compared to MCU-1 and MCU-3. A similar pattern is observed in the prediction speeds of the models in Table 5. However, the overall training times are relatively long. Even the model with the lowest number of parameters, Model-1, requires up to 12 seconds to complete training on 100 samples. In contrast, Model-3, with the highest number of parameters, takes up to 135.52 seconds to finish training on the MCU with the highest clock speed. Therefore, training ANN models directly on MCUs using sequential programming is not feasible, making it challenging to ensure real-time performance on the devices. Conversely, the prediction speeds of the ANN models can range from approximately 1.7 Hz to 23 Hz on high-clock-speed MCUs. This suggests that the trained models can be applied to MCUs for prediction tasks, but retraining them is difficult due to the lengthy interruption caused by the device's training process.

Table 4. Evaluate training time

Name	Training time on PC and MCU (s)		
	Model-1	Model-2	Model-3
MCU-1	12.00	76.64	149.60
MCU-2	27.51	217.64	401.31
MCU-3	16.86	76.96	135.52
MCU-4	13.64	215.62	417.61
PC	0.11	0.85	1.57

Table 5. Evaluate predict time

Name	Inference time on PC and MCU (s)		
	Model-1	Model-2	Model-3
MCU-1	0.047	0.357	0.660
MCU-2	0.093	1.221	2.247
MCU-3	0.043	0.322	0.594
MCU-4	0.074	1.111	2.137
PC	5e-4	33e-4	58e-4

The MSE results between PC and MCUs are presented in Figure 6, demonstrating a significant similarity between the two platforms with only a tiny difference of approximately 0.1. The main reason for this slight discrepancy is that MCUs support data representation and operations with lower precision than one PC. Nevertheless, this proves that trained ANN models can be directly deployed on resource-constrained devices. Therefore, a parallel programming mechanism needs to be implemented to achieve simultaneous training and prediction on these MCUs.

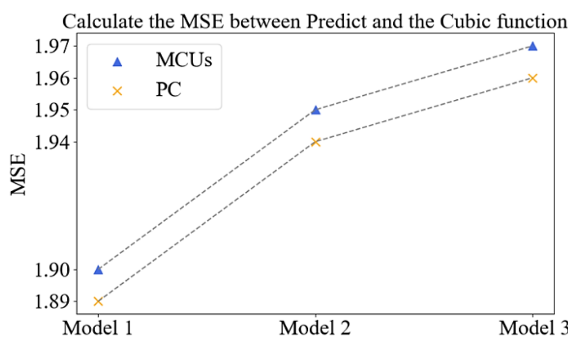

Figure 6. Calculate MSE between predict and cubic function

3.2. Dual-core performance on micro-controller units

Figure 7 presents the results of parallel training and prediction on Dataset-2. The results show that, except for the case of training the ANN models for the first time, training and prediction occur in parallel in subsequent training sessions. This addresses the issue of interrupting prediction during training in sequential programming. Dataset-2 is divided into four subsets to perform sequential training, so the model's loss values gradually decrease over the training sessions, as illustrated in the chart. Indeed, after completing the six training sessions, the average loss value is in the range of 0.02, and the MSE of prediction is 0.04, representing a significant improvement compared to previous training sessions. This is particularly suitable for supervised learning applications on embedded systems based on real-time sensor data.

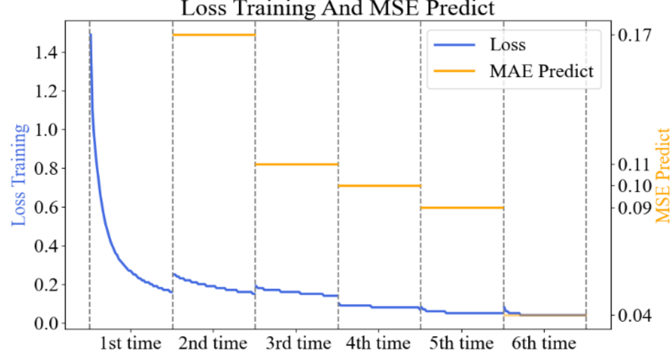

Figure 7. Experimental results of parallel training and prediction on MCU-1

Figure 8 presents the prediction results of model-1 after five training sessions, where the lines represent the prediction results of the model with the test dataset of Dataset-2. The results show that, after each training session, the prediction function gradually approaches the original graph of the cubic function. The prediction MSE decreases from 0.17 to 0.04 after five training sessions, indicating a 76.5% improvement in the model-1 MSE. This training process can continue to improve the accuracy of the model further because the device's prediction is not interrupted by the training process. The parameters will be updated after each training session, almost without affecting the prediction. In summary, although the training time increases with the complexity of the model, this drawback has been overcome with the multi-task programming technique running on multi-core MCUs. This demonstrates that the ANN model library and multi-task algorithm can be applied to deploy traditional ANN models directly on common MCUs.

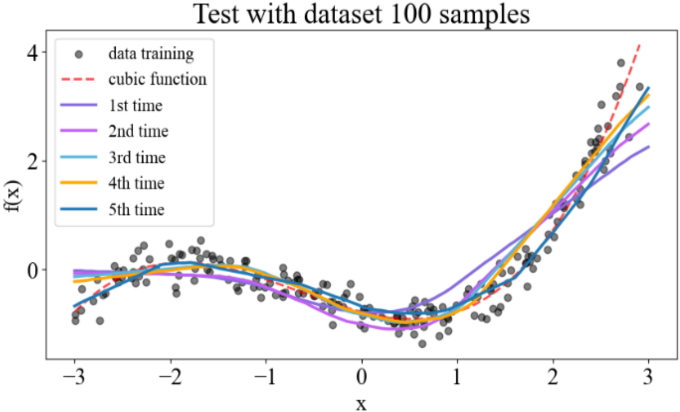

Figure 8. Prediction results on MCU-1

4. DISCUSSION

In this study, we developed a library for deploying ANN models directly on MCUs. The created models are trained and predicted on the MCUs. Experimental results show that the prediction time of the models after training is relatively fast. However, the training time could be longer, making it impractical to deploy traditional ANN models directly on MCUs due to the lengthy training process, which makes it challenging to ensure real-time performance. The application of multi-task programming based on freeRTOS has addressed this drawback. The training process can be iterated to improve the model's accuracy while the prediction process continues continuously with the latest parameter set, ensuring the system's functionality.

By applying multi-task programming to enable parallel training and prediction processes, this research allows the direct implementation of traditional ANN models on MCUs without optimizing algorithms. This is advantageous because the relevant mathematical operations have been well-established and thoroughly evaluated. The developed ANN library is highly flexible, allowing easy addition and modification of layers, activation, and loss functions through the library's application programming interface (API) functions. Moreover, this ANN library has the potential for scalability and customization across various platforms. Indeed, in addition to the integrated activation and loss functions, new functions can be easily programmed using their mathematical representations. The library is written in the object-oriented programming language C++ and focused on core features, making it easy to transition to other programming languages or platforms in the future. It can also be customized to be deployed on various multi-core MCU models. Compared to the Train++ algorithm [20], both studies support multiple MCUs. However, this research demonstrates greater flexibility in implementing various ANN models with different structures, whereas Train++ does not support hidden layers.

Despite the advantages, the ANN library in this study also has three main limitations. First, optimization algorithms have yet to be applied to accelerate processing speed, posing challenges when applied to applications demanding high computational speeds and making it difficult to implement complex-scaled models due to the limited memory of MCUs. Second, the level of multitasking could be higher, utilizing only two main parallel tasks and not fully exploiting the computational capabilities of multi-core MCUs. Task partitioning for training and prediction should be prioritized and concentrated on in the future. Finally, the library is currently limited to ANNs, and integrating data storage capabilities may prove challenging when applied to supervised learning-based classification applications.

In reality, the ANN model cannot only handle regression problems but can also be extended to perform various tasks, such as classification and anomaly detection. However, implementing anomaly detection based on unsupervised learning methods would be more feasible, and storing labeled data on the MCU is complex. This research can be highly applicable in real-world scenarios, especially in IoT applications where devices are being explored to integrate self-learning, analysis, and decision-making capabilities locally instead of relying on an AI network operating on the system's cloud server.

5. CONCLUSION

In this study, a library has been developed to deploy ANN models directly on multi-core MCUs. The models, once created, are trained and predicted on the MCUs. Experimental results demonstrate that the prediction time for the models after training is relatively fast. In contrast, the shortest training time takes up to 12 seconds for the most complex model. However, this issue has been addressed by multitasking using

freeRTOS on multi-core MCUs, allowing the training and prediction processes to occur concurrently without interference. Furthermore, the training process can continue to improve the accuracy of the ANN model when additional relevant data is collected. This research can potentially integrate ANNs into embedded devices, especially in the IoT domain. For example, when estimating irrigation needs in agriculture based on soil moisture, soil temperature, and air temperature to determine the amount of irrigation water, the edge device will be retrained regularly to adapt to climate conditions.

REFERENCES

[1] R. S. -Iborra and A. F. Skarmeta, "TinyML-enabled frugal smart objects: challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020, doi: 10.1109/MCAS.2020.3005467.
[2] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A Comprehensive survey on TinyML," *IEEE Access*, vol. 11, pp. 96892–96922, 2023, doi: 10.1109/ACCESS.2023.3294111.
[3] Y. Y. Siang, M. R. Ahamd, M. Shafinaz, and Z. Abidin, "Anomaly detection based on tiny machine learning: A review," *Open International Journal of Informatics*, vol. 9, no. Special Issue 2, pp. 67–78, 2021.
[4] B. Sun, S. Bayes, A. M. Abotaleb, and M. Hassan, "The case for tinyML in healthcare: CNNs for real-time on-edge blood pressure estimation," in *Proceedings of the ACM Symposium on Applied Computing*, in SAC '23. ACM, 2023, pp. 629–638, doi: 10.1145/3555776.3577747.
[5] D. L. Dutta and S. Bharali, "TinyML Meets IoT: a comprehensive survey," *Internet of Things (Netherlands)*, vol. 16, 2021, doi: 10.1016/j.iot.2021.100461.
[6] N. Schizas, A. Karras, C. Karras, and S. Sioutas, "TinyML for ultra-low power AI and large scale IoT deployments: a systematic review," *Future Internet*, vol. 14, no. 12, 2022, doi: 10.3390/fi14120363.
[7] P. Warden and D. Situnayake, *TinyML: machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*, Sebastopol, USA: O'Reilly Media, Inc, 2019.
[8] J. Lee and H.-J. Yoo, "An overview of energy-efficient hardware accelerators for on-device deep-neural-network training," *IEEE Open Journal of the Solid-State Circuits Society*, vol. 1, pp. 115–128, 2021, doi: 10.1109/ojsscs.2021.3119554.
[9] O. D. Incel and S. O. Bursa, "On-device deep learning for mobile and wearable sensing applications: A review," *IEEE Sensors Journal*, vol. 23, no. 6, pp. 5501–5512, 2023, doi: 10.1109/JSEN.2023.3240854.
[10] D. Nadalini, M. Rusci, L. Benini, and F. Conti, "Reduced precision floating-point optimization for deep neural network on-device learning on microcontrollers," *Future Generation Computer Systems*, vol. 149, pp. 212–226, 2023, doi: 10.1016/j.future.2023.07.020.
[11] L. Wu, J. Liu, S. Vazquez, and S. K. Mazumder, "Sliding mode control in power converters and drives: a review," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 3, pp. 392–406, 2022, doi: 10.1109/JAS.2021.1004380.
[12] A. N. Mazumder *et al.*, "A survey on the optimization of neural network accelerators for micro-AI on-device inference," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 532–547, 2021, doi: 10.1109/JETCAS.2021.3129415.
[13] J. Lin, L. Zhu, W. M. Chen, W. C. Wang, C. Gan, and S. Han, "On-device training under 256KB memory," *Advances in Neural Information Processing Systems*, vol. 35, 2022.
[14] M. Chowdhary and S. S. Saha, "On-sensor online learning and classification under 8 KB memory," in *2023 26th International Conference on Information Fusion, FUSION 2023*, IEEE, 2023, pp. 1-8, doi: 10.23919/FUSION52260.2023.10224228.
[15] E. L. Lamie, "Real-time embedded multithreading using threadX and MIPS," *Real-Time Embedded Multithreading Using ThreadX and MIPS*. CRC Press, pp. 1–465, Apr. 2019, doi: 10.1201/9780429187858.
[16] R. V. Aroca and G. Caurin, "A real time operating systems (RTOS) comparison," *WSO - Workshop de Sistemas Operacionais*, vol. 12, pp. 2441-2452, 2009.
[17] A. Thomas, "Enclaves in real-time operating systems," M.Sc. Dissertation, Department of Electrical Engineering and Computer Science, University of California, Berkeley, USA. 2021.
[18] M. Z. H. Zim, "TinyML: analysis of xtensa LX6 microprocessor for neural network applications by ESP32 SoC," *ArXiv-Computer Science,* pp. 1-6, 2021, doi: 10.13140/RG.2.2.28602.11204.
[19] V. K. Nguyen, V. K. Tran, H. Pham, V. M. Nguyen, H. D. Nguyen, and C. N. Nguyen, "A multi-microcontroller-based hardware for deploying Tiny machine learning model," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 5, pp. 5727–5736, 2023, doi: 10.11591/ijece.v13i5.pp5727-5736.
[20] B. Sudharsan, P. Yadav, J. G. Breslin, and M. Intizar Ali, "Train++: an incremental ML model training algorithm to create self-learning IoT dDevices," in *Proceedings - 2021 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Internet of People, and Smart City Innovations, SmartWorld/ScalCom/UIC/ATC/IoP/SCI 2021*, IEEE, 2021, pp. 97–106, doi: 10.1109/SWC50871.2021.00023.
[21] M. Craighero, D. Quarantiello, B. Rossi, D. Carrera, P. Fragneto, and G. Boracchi, "On-device personalization for human activity recognition on STM32," *IEEE Embedded Systems Letters*, vol. 16, no. 2, pp. 106-109, June 2024, doi: 10.1109/LES.2023.3293458.
[22] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. E. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, Nov. 2018, doi: 10.1016/j.heliyon.2018.e00938.
[23] A. D. Jagtap and G. E. Karniadakis, "How important are activation functions in regression and classification? a survey, performance comparison, and future directions," *Journal of Machine Learning for Modeling and Computing*, vol. 4, no. 1, pp. 21–75, 2023, doi: 10.1615/jmachlearnmodelcomput.2023047367.
[24] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, Berlin, Heidelberg: Springer, 2012, pp. 421–436, doi: 10.1007/978-3-642-35289-8_25.
[25] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation," *PeerJ Computer Science*, vol. 7, pp. 1–24, 2021, doi: 10.7717/PEERJ-CS.623.
[26] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: modeling the costs of mislabeling," *IEEE Access*, vol. 8, pp. 4806–4813, 2020, doi: 10.1109/ACCESS.2019.2962617.

BIOGRAPHIES OF AUTHORS

Bao-Toan Thai is a B.S. degree student in Automation and Control Engineering of the Faculty of Automation Technology, College of Engineering, Can Tho University, Vietnam. He can be contacted at email: thaibaotoan.ag@gmail.com.

Vy-Khang Tran is a MSc degree student in Automation and Control Engineering of the Faculty of Automation Technology, College of Engineering, Can Tho University, Vietnam. His research interests focus on embedded systems and AIoT- and IoT-based applications in environmental and agricultural control. He can be contacted at email: tranvykhang1906@gmail.com.

Hai Pham received his master's degree from University of South Australia (UniSA) in 2010. Since 2012, he has been a lecturer at Faculty of Automation Technology, College of Engineering, Can Tho University. His research interests focus on Bistatic LIDAR system for gas measurement in environmental and agricultural applications. He can be contacted at email: ptlhai@ctu.edu.vn.

Chi-Ngon Nguyen received B.S. and M.S. degrees in Electronic Engineering from Can Tho University and the National University, Ho Chi Minh City University of Technology, Vietnam, in 1996 and 2001, respectively. The degree of Ph.D. in Control Engineering was awarded by the University of Rostock, Germany, in 2007. Since 1996, he has worked at the Can Tho University. He is an associate professor in automation at Faculty of Automation Technology, and former dean of the College of Engineering at the Can Tho University. Currently, he is a Vice Chairman of the Board of Trustee of Can Tho University. His research interests are intelligent control, medical control, pattern recognition, classifications, speech recognition, computer vision and agricultural automation. He can be contacted at email: ncngon@ctu.edu.vn.

Van-Khanh Nguyen received his master's degree from Ho Chi Minh University of Technology, Vietnam, in 2014 and his Doctor of Engineering degree from Tokyo University of Marine Science and Technology, Japan, in 2020. Since 2007, he has been a lecturer at the Faculty of Automation Technology, College of Engineering, Can Tho University. Currently, he is the head of PLC Technology and Industrial IoT Lab. His research interests concentrate on embedded systems and AIoT- and IoT-based applications in environmental and agricultural control, electrocardiogram (ECG) real-time classification, and anomaly detection. He can be contacted at email: vankhanh@ctu.edu.vn.