Development Of ARMA Algorithm For System Identification In Structure Health Monitoring

Faizal Afzal[1], M. Irfan Anis[2], Marium Shakeel[2]

***Abstract*— Amplitudes, peaks shift and waveforms altered the signal response when structural damage severity and position change, thus having a strong association between damage cases and signal response shapes. The damage detection and building structural state evaluation are tough to comprehend. As the number of structural failures has increased, the development of methods for recognizing the degradation become increasingly important. Structural health of buildings and critical infrastructure should be monitor for signs of deterioration or impending failure. Time series modeling and forecasting was an effective tool for structural analysis. To emulate via artificial neural networks the behavior of time series model, the Autoregressive with Moving Average (ARMA) represents one of the most effective methods for structural characterization in operative environments. The novelty of this work is to attain using the unique TinyML approach that allowed for the embodiment of the developed models into a resource-constrained device. The proposed models were trained and tested using data collected by sensors strategically placed on engineering structures. The identical models were then converted to the TFLite format that are designed for devices with limited resources. The converted models were tested using the same dataset on Arduino and STM32 boards to assess how they perform in real-world scenarios. For univariate time series forecasting an Artificial Neural Network (ANN) method reduce computational effort and reduced severity outperformed Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN).**

***Keywords*— Degree of Freedoms, Structural Health Monitoring, Tiny Machine Learning, TensorFlow Lite, Neural Network**

INTRODUCTION

The research community has shown a growing interest in the identification of damage and assessment of the condition of building structures in recent decades due to long-term deterioration and exposure to extreme events such as earthquakes, resulting in many structures nearing the end of their life cycle. The absence of proper retrofitting measures can cause buildings to partially or completely collapse without warning, leading to human fatalities and significant economic loss and repercussions. Therefore, it is essential to conduct damage detection and condition evaluation of building structures throughout their lifespans, especially for older structures or those suspected of having endured excessive loads [1]. The frequency of building and bridge collapses with little or no warning has been on the rise in many regions around the world. Consequently, the development of techniques for detecting the degradation or damage resulting from such incidents has become increasingly critical. Similar to monitoring the condition of a patient in a hospital, buildings and essential infrastructure should be regularly assessed for signs of deterioration or imminent impairment that may lead to collapse [16].

The aim of the proposed solution is to integrate Structural Health Monitoring (SHM) systems into preventive maintenance procedures, particularly for newly constructed civil structures. By adopting SHM, not only can the safety of existing civil structures be enhanced, but it can also furnish more efficient tools for the preventive maintenance of future ones. SHM is an inclusive framework for diagnosing, analyzing, and forecasting damage to infrastructure in aerospace, civil, and mechanical engineering. It encompasses the strategic installation of various sensors on civil structures, including strain gauges and temperature sensors to gauge strain on columns and roads, barometers and hydrometers to assess the impact of rain on the structure, and accelerometers to facilitate vibration-based analysis and diagnostics [2]. In addition, the utilization of real-time sensing technology enables the collection of structural health data without disrupting the normal flow of traffic on the bridge. Prior to the adoption of SHM, the only available options for diagnosis were to wait for noticeable signs of structural decay or to conduct routine maintenance.

In this study, accelerometer sensors are utilized to assess the health of vibrating structures in consideration of their external surroundings. However, real-time sensing with accelerometers for vibration-based diagnostics [3] requires the collection of numerous parameters that can be expensive in terms of device battery life, network bandwidth, and data transmission time to the data center, hindering the adoption of machine learning. To overcome this, machine learning is performed entirely in the cloud. Time series modeling is a possible method of structural characterization that can provide critical information about the structure's health condition. Therefore, this paper focuses on implementing system identification through neural network models and ARMA, where the estimation of the model order is a challenging task. The novelty of this work is the use of artificial intelligence to solve problems related to time series forecasting, including model order identification and forecasting based on the previously estimated quantity. Additionally, to

[1]Università di Bologna, Italy
[2]Iqra University, Karachi
Email: * mirfananis@iqra.edu.pk

demonstrate the practicality of these processing tools in real-world application scenarios, the implemented models are tested on resource-constrained devices.

This research study employed an ANN structure to ascertain the optimal model order for ARMA modeling of linear, time-invariant systems based on the system's input and output data. The error loss function for the neural network, or coefficient of determination, was used for this purpose. This ANN-based SHM ML model was generated with data collected at a sampling frequency of 2^{12} and 4KB samples (50Hz) for a 6-degree-of-freedom system [4]. The data used in this study was sourced from the "Numerical tool for the Simulation of Vibration Data in Civil Infrastructures". The primary objective of this research project was to showcase the practicality of vibration-based engineering techniques for structural diagnostics and testing aimed at improving failure prevention [4]. This research study utilized data obtained from multiple strategically positioned acceleration sensors on engineering structures to train unique TinyML models. These models underwent rigorous training and testing to assess their performance in terms of precision and accuracy, using the Tensorflow (TF) and Keras frameworks [5]. The same models were then converted to the TFLite format, which is optimized for microcontrollers and other resource-constrained devices. The transformed models were evaluated on Arduino and STM32 boards using the same dataset to determine their performance in real-world scenarios. The primary parameters of the TFLite model and the ported models on Arduino and STM32 were computed and verified against the original TF version [17].

In the context of always-on detection, TinyML is often used in the initial stage. This approach involves performing simple processing locally on the device, rather than sending a continuous stream of data to the cloud and consuming significant amounts of Internet bandwidth [19].

The stated models were implemented and run on an Embedded Systems based platform [6]. Performing inference on a cloud-based platform would necessitate regular transmission of data from remote sensors [20]. This would lead to inference latency, impeding the real-time functionality of the system, higher bandwidth consumption, and increased power usage by the sensor node due to data transmission. Conversely, performing inference on an edge device eliminates the need for data transmission, resulting in minimal diagnosis latency. Additionally, performing inference on the edge device consumes less power, leading to longer battery life, and communication needs are less frequent compared to data transmission.

This research paper aims to implement novel TinyML models in Structural Health Monitoring applications by means of an Artificial Intelligence approach utilizing ARMA modelling. This includes deployment of three types of Neural Networks (NNs) on both an Arduino Nano 33 BLE Sense board and an STM32 Nucleo-144 development board. The use of ANNs for damage detection offers several appealing features, including the ability to automate the problem diagnosis procedure once the network has been trained. However, when dealing with structures with multiple degrees of freedom, ANNs often need a large amount of computational labor. The most ANN-based damage detection applications are restricted to compact structures with few degrees of freedom [7]. Identification procedures often involve costly calculations, making it impractical to identify all unknown characteristics, such as mass and stiffness, simultaneously from input-output data, particularly for structures with numerous unknowns. Additionally, established system identification methodologies for simple structures must be evaluated for real-world infrastructures [18], which are more flexible and have greater degrees of freedom (DOFs). To reduce the number of unknowns and improve measurement and identification accuracy, it is advantageous to divide large-scale structures into multiple smaller substructures.

The article is organized as follows: Section II outlines the proposed artificial neural network (ANN) structure. Section III describes the deployment of TinyML models, while Section IV presents the experimental results. Finally, the results are discussed along with the applicability and limitations of the approach.

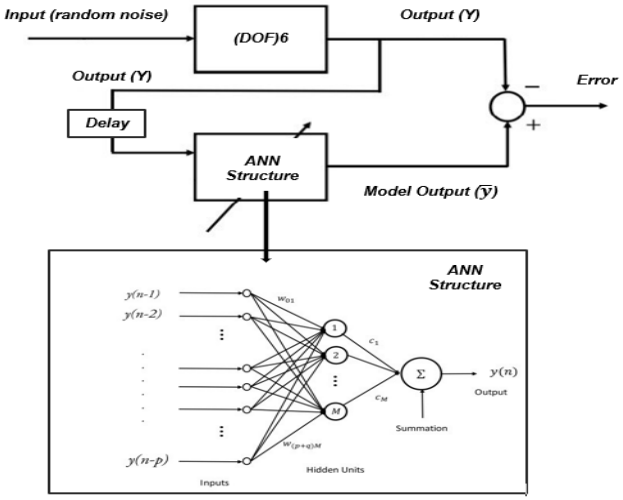


Figure. 1: Proposed ANN structure

The figure 1, presented in this paper depicts the block structure of the system identification process. The algorithm is applied to a 6-degree-of-freedom structure in order to determine a model for the unknown system. The training of the network aims to emulate the behavior of the system and conduct a system identification process. Once the training is complete, the system can be characterized by determining the ARMA parameters.

I. MODEL TINYML DEPLOYMENT

2.1 The Autoregressive Moving Average (ARMA) Model

An ARMA (p, q) model is a hybrid of AR(p) and MA(q) models that can be used to model univariate time series. The future

value of a variable is supposed to be a linear combination of p prior observations, a random error, and a constant term in an AR(p) model. The AR(p) model is expressed mathematically as follows [8]:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \varepsilon_t$$
$$= c + \phi_1 y_t - 1 + \phi_2 y_t - 2 + \cdots \ldots \ldots + \phi_p y_t - p + \varepsilon_t$$

y_t and ε_t are the actual value and random error (or random shock) at time t, ϕi (i = 1, 2..., p) are model parameters, and c is a constant, respectively. The integer constant p represents the model's order. The constant word is occasionally eliminated for the purpose of simplicity. The Yule-Walker equations are commonly used to estimate parameters of an AR process from a time series [21]. Previous errors are used as explanatory variables in an MA(q) model, similar to how an AR(p) model regresses against previous series values. This is how the MA(q) model is defined:

$$y_t = \mu + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t = \mu + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots \ldots \ldots \ldots + \theta_g \varepsilon_{t-q} + \varepsilon_t$$

The mean of the series is, the model parameters are j (j = 1, 2..., q), and the model order is q. The random shocks are a white noise process, that is a sequence of independent and identically distributed (i.e.) random variables with a constant variance of 2. Random shocks are believed to follow a standard normal distribution in most cases. A moving average model, then, is a linear regression of the current time series observation against random shocks from one or more previous observations. Fitting an MA model to a time series is more challenging than fitting an AR model the random error terms in the former are not anticipated [9].

2.2 Order selection

This study proposes the use of an artificial neural network (ANN) approach to identify the optimal order of an autoregressive-moving average (ARMA) model for linear and time-invariant systems based on given input and output data. The selection of the ARMA model order is accomplished through the minimization of a neural network error loss function, namely the coefficient of determination.

In a linear regression context, R^2 is a statistic that measures a model's ability to predict or explain a result. The amount of variance in the dependent variable (Y) predicted or explained using linear regression and the predictor variable is denoted by R^2 (X, also known as the independent variable [10].

A high R^2 score indicates that the model is a good fit for the data, though fit interpretations differ depending on the study environment. An R^2 of 0.35, for example, suggests that using the covariates in the model to predict the outcome may explain 35% of the variation in the outcome. That proportion of variance may be difficult to predict in certain disciplines, such as the social sciences; in others, such as the physical sciences, one would expect R^2 to be much closer to 100%. In theory, R^2 must be at least 0. Because linear regression is based on the best possible fit, R^2 will always be greater than zero, even if the predictor and result variables have no relationship.

Though the new predictor is unrelated to the outcome, R^2 increases when it is added to the model. The adjusted, R^2 (typically depicted by a bar over the *R* in R^2) contains the same data as the normal R^2, but penalizes for the number of predictor variables in the model to account for this effect. The result, R^2 increases as additional predictors are added to a multiple linear regression model, but the adjusted R^2 only increases if the increase in R^2 is more than what would be expected by chance alone. The adjusted R^2 in such a model is the most realistic estimate of the proportion of variance predicted by the covariates in the model. Each of the n variables $y_1, \ldots\ldots, y_n$ (collectively known as y_i) in a data set is associated with a fitted (or modeled or predicted) value $f_1, \ldots\ldots, f_n$ (known as fi).

Define the residuals as $e_i = y_i - f_i$ (forming a vector $\boldsymbol{e}$).

If $\bar{y}$ is the mean of the observed data:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

then the data set's variability can then be calculated using two sums of squares formulas:

The residual sum of squares, often known as the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

The most general definition of the coefficient of determination is.

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

In the best situation, the modeled and observed values are identical, resulting in $SS_{res} = 0$ and $R^2 = 1$. $R^2 = 0$ is the value for a baseline model that always predicts $\bar{y}$. Models with lower prediction accuracy than this baseline have a negative R^2.

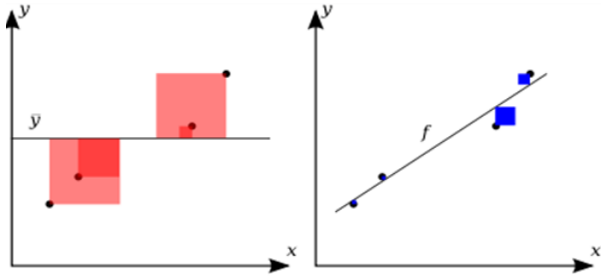


Figure. 2: $R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$

The degree of fit of the linear regression (shown on the right) to the data relative to the simple average (on the left) can be evaluated by examining the value of $\boldsymbol{R^2}$. The areas of the blue squares represent the squared residuals for the linear regression. In Figure 2, the squared residuals related to the average value are depicted by the areas of the red squares.

II. ARMA Model Order Estimation

For a given input and output time series data, the neural networked structure is trained for a specific number of epochs using a sequence of different orders (1 p pmax, 0 q qmax). Depending on the order combination, the ANN structure clearly illustrates that the input layer employs unique lags of the input and output time series, that are defined by the AR and MA order. The mean square error (MSE) of the neural network for the correct model order should be as low as possible. Higher order combinations and noise corruption can still reduce the loss value, but at a much slower rate. The effective order selection with the least MSE loss outperforms estimate in some circumstances [11]. To solve this challenge, by determined the ideal model order using both the loss value and the rate of change of loss. For each combination of model orders, by calculated the average of the initial MSEs by sorting the MSE values of distinct epochs in ascending order.

The loss function's order is modest since the lowest model order (p) is defined as the right model order. There are a variety of situations in which the loss order (r value) is minimum for only a few (one or two) or most of the model orders combinations. The right model orders in those conditions were the lowest model orders with the lowest loss value. By looking at the output error loss of the ANN structure at different orders and discovered that the best projected output was when the order was set to p = 6 with a loss value of 0.28, as shown in figures 3.

Figure 3: loss function R^2

Figure 3 figure shows loss function R^2 for different orders using the system identification structure with proposed ANN topology.

The optimal model order (p) is defined as the correct model order. The best results were assessed at order p=6 with a loss value of 0.047 as the minimum.

In this study, the potential for Neural Network models to emulate the system identification capability of an ARMA model was explored. Three types of Neural Networks (NNs) were analyzed to predict the behavior of the system identification: A Convolutional Neural Network (CNN) represented in a block structure in Figure 5a, a Recurrent Neural Network (RNN) block representation shown in Figure 5b, and an Artificial Neural Network (ANN) block structure depicted in Figure 5c. The predicted most satisfactory behavior was achieved through these analyses.

```
#ANN Model
model = tf.keras.Sequential()
# First layer takes a scalar input and feeds it through 64 "neur
# neurons decide whether to activate based on the 'relu' activat
model.add(keras.layers.Dense(64, activation='relu', input_shape=
# The new second layer will help the network learn more complex
model.add(keras.layers.Dense(64, activation='relu'))
# Final layer is a single neuron, since we want to output a sing
model.add(keras.layers.Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse', metrics=[r2_metric])
#model.compile(optimizer='adam', loss="mse", metrics=["mae"])
model.summary()

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 64)                448
_________________________________________________________________
dense_7 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 65
=================================================================
Total params: 4,673
Trainable params: 4,673
Non-trainable params: 0
```

(a)

```
#RNN Model
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, LSTM, GRU, SimpleRNN,
from tensorflow.keras.models import Sequential, Model
# one layer SimpleRNN seems to be enough for this data set
model = Sequential()
model.add(SimpleRNN(128, activation='relu', input_shape=(n_steps_
# combines final outputs from RNN into continous output
model.add(Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse', metrics=[r2_metric])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 128)               16640
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 16,769
Trainable params: 16,769
Non-trainable params: 0
```

(b)

```
#CNN Mdoel
# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', i
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse', metrics=[r2_metric])
```

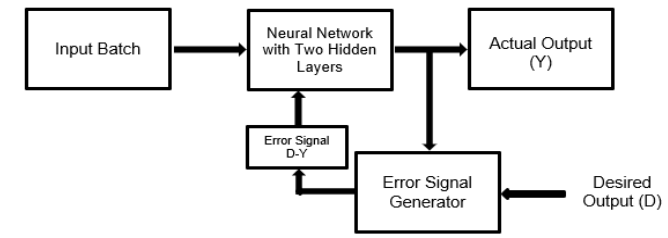
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 19, 64)            192
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 9, 64)             0
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 37,185
Trainable params: 37,185
Non-trainable params: 0
```

(c)

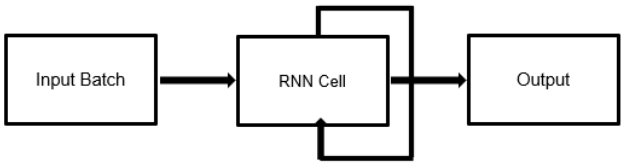
Figure 4: Model's Parameters. (a) ANN Model, (b) RNN Model, (c) CNN Model

Furthermore, this system necessitates that the same inputs and output shapes be supplied as those used with Artificial Neural Networks (ANNs). Through the process of sampling, the coefficients remain unchanged and ready for the next system of signal reception.

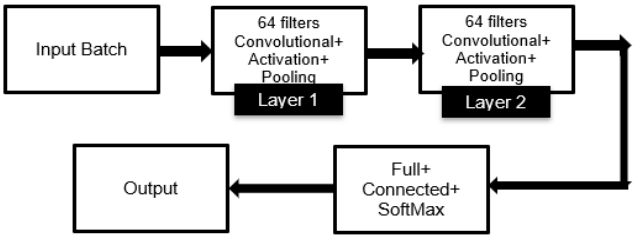
Figure 4a shows that the ANN model not only has the most satisfactory behavior, but also has a lower total number of parameters than CNN and RNN structures. Figure 4b illustrates the computation complexity of the CNN structure. For comparison with other structures, the total number of parameters in the RNN structure [12] shown in figure:4c.



(a)



(b)



(c)

Figure 5: Block structure of (a) ANN Model, (b) RNN Model, (c) CNN Model

III. Results

Figure:6 shows the DOF 6 structure that can be enhance condition or damage condition in both cases having different modelling output by tracking these changes can infer the health of structure. Collected signals from numerical tool (DOF)6 data. Each signal sized 4KB samples at frequency 100Hz.

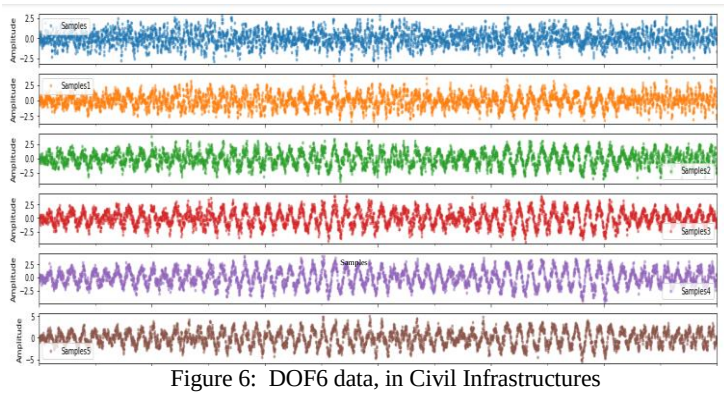
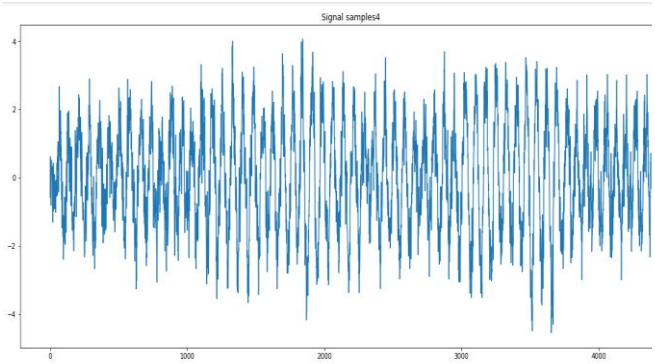
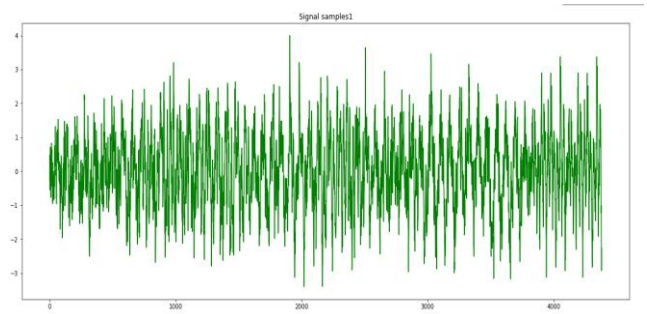


Figure 6: DOF6 data, in Civil Infrastructures

To assess the performance of a model, by looking at how it performs on its training data. However, this only tells us part of the story. During training, a model learns to fit its training data as closely as possible. The figure:7a showing samples of training signal used in teaching our model.



(a)



(b)

Figure7: Sample sequence (a) 4 and (b) 1 for testing ANN model

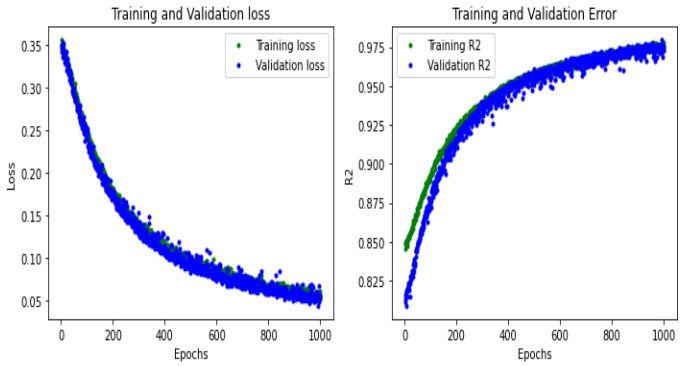
In certain circumstances, the model may overfit the training data, leading to good performance on training data but poor performance in real-world situations. To detect this, the model should be validated with new data that was not used during training. The data should be split into chunks for training and validation in such a way that they have the same information distribution and preserve the data structure. For time-series data, the chunks can be contiguous in time, while for non-time-series data, the data points can be sampled randomly. During training, the model learns from the training dataset, and validation and test datasets are used periodically to calculate loss. Since the model has not seen this data before, its loss score is a more reliable measure of its performance. By comparing training and test loss (and other metrics), we can determine whether the model is overfitting. Figure 7b shows the use of Sample Sequence 1 to evaluate a model's performance.

Figure 8 illustrates the split and shape of the univariate sequence for both the training and testing datasets. The order of the ARMA model, which is 6, was determined based on the minimum loss function, specifically the coefficient of determination, as explained in the order selection section.

```
[ ] #@title Prediction from n past values
    n_steps_in = 6 #@param {type:"slider", min:1, max:100, step:1}
    n_steps_out = 1

[ ] X_train, Y_train = split_sequence(samples, n_steps_in, n_steps_out)
    X_train.shape, Y_train.shape

    ((4377, 6), (4377, 1))

    X_test, Y_test = split_sequence(test, n_steps_in, n_steps_out)
    X_test.shape, Y_test.shape

    ((4377, 6), (4377, 1))
```

Figure 8: Splitting and Shaping, how much samples of past values are using to predict next value.

To stop training when a model's performance stops improving. At the point that it begins to make accurate predictions, it is said to have converged. To regulate whether a model has coincided, by inspecting graphs of its show during training. Two ordinary execution metrics are loss and accuracy [13]. The loss metric provides a quantitative measure of the model's deviation from the expected results, while the accuracy metric indicates the percentage of correct forecasts using the coefficient of determination method. An ideal model should have a loss of 0.0 and accuracy of 100%, but real-world models are seldom perfect. In Figure 9a and b, the loss and accuracy are shown during the training of a deep learning network. As training progresses, the accuracy increases and the loss decreases, until the model reaches a point where it no longer improves. To attempt to improve the model's performance, we can modify our model architecture and adjust various hyperparameters that regulate the training process, such as the number of teaching epochs and the number of neurons in each layer.



(a) (b)

Figure 9: (a) Loss vs. epoch, (b) Accuracy vs. epoch

Based on the Loss vs. epoch (blue validation, green training) graph in Figure 9 (a), and the Accuracy vs epoch (blue validation, green training) graph in Figure 9 (b), it can be inferred that the model is currently in a balanced state with high accuracy.

The loss and accuracy during training for an ANN model, as training progresses, accuracy increases and loss is reduced, until by reaching a point at which the model no longer improves.

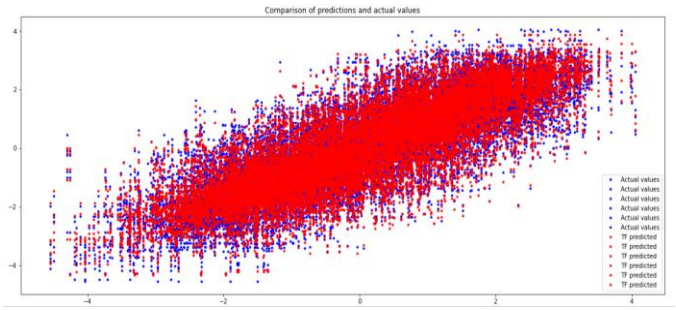


Figure 10: Eye Diagram

Figure 10 illustrates the Artificial Neural Network (ANN) approach that was developed to estimate and train the ARMA model. The predicted outcomes are compared to the actual samples using our test dataset.

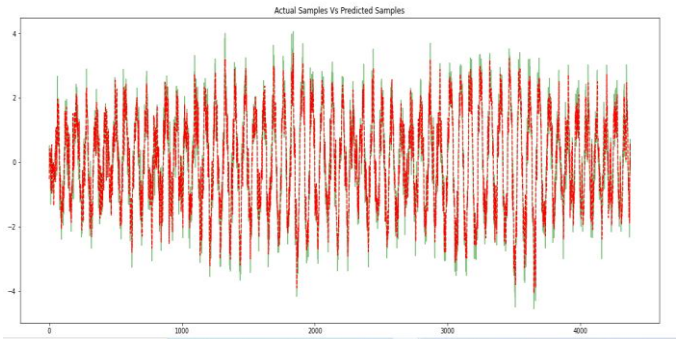


Figure 11: Comparison of expected time domain values with actual values

A comparison between the expected time domain values and the actual values is illustrated in Figure 11. The red signal indicates the predicted results after ARMA model training, while the green signal represents the actual system samples.

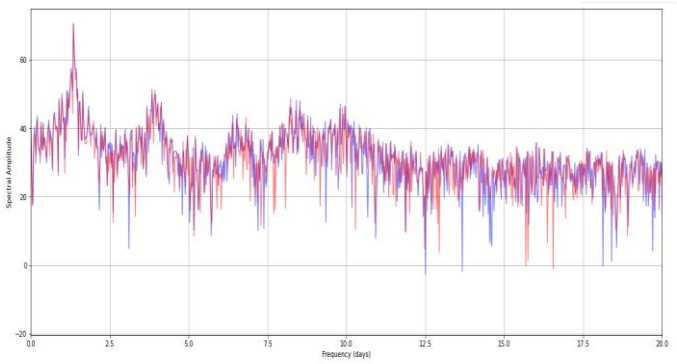


Figure 12: Shows frequency response (blue represents original signal and red is predicted signal)

TensorFlow	163860 bytes	
TensorFlow Lite	20328 bytes	(reduced by 143532 bytes)
TensorFlow Lite Quantized	6912 bytes	(reduced by 13416 bytes)

Figure 13: ARMA model size using different approach

The size of the ARMA model has been reduced using the quantized approach and integrated with Tensor Flow Lite for deployment on embedded devices. The original size of the model was 163860 bytes, but after quantization, it has been reduced to 6912 bytes, as illustrated in Figure 13.

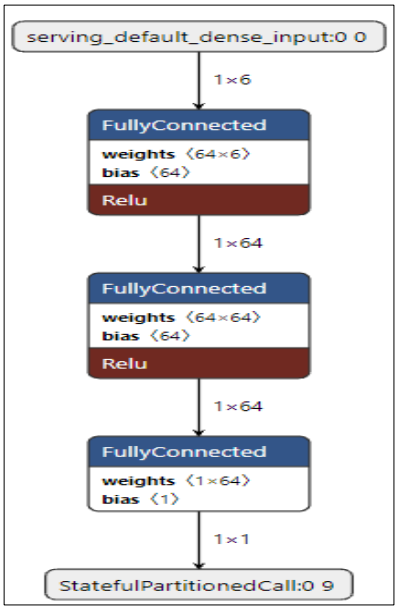


Figure 14: TFLite Model visualized in Netron

Figure 14 depicts the deployment of the TFLite model, visualized in Netron, on both the Arduino Nano BLE 33 and STM32 Nucleo F7 Board, along with the use of the same dataset for testing the model.

The use of a quantized version of Tensor Flow Lite, as shown in Figure 15, allowed for a simplified model that was converted to a binary file, reducing its size to 6912 bytes. This reduced model was deployed on an embedded device, specifically an Arduino Nano BLE sensor. The size of the Tensor was 21KB, while the script used for inference was around 30KB in size. The deployed model was tested, and the recorded results can be seen in Figure 14, along with the frequency response depicted in Figure 16. A comparison was made between the time domain signal predicted by the Colaboratory file and the data collected from the embedded device [14] is shown in the result.

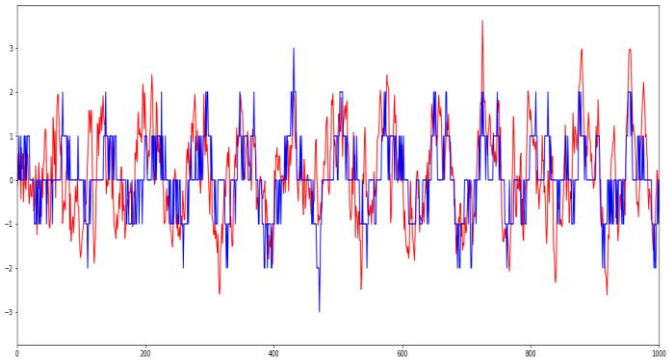


Figure 15: Frequency signal recorded

In Figure 15, the blue line represents the frequency signal recorded from the Nano BLE sensor, while the red line shows the predicted signal generated by the ANN model after training.

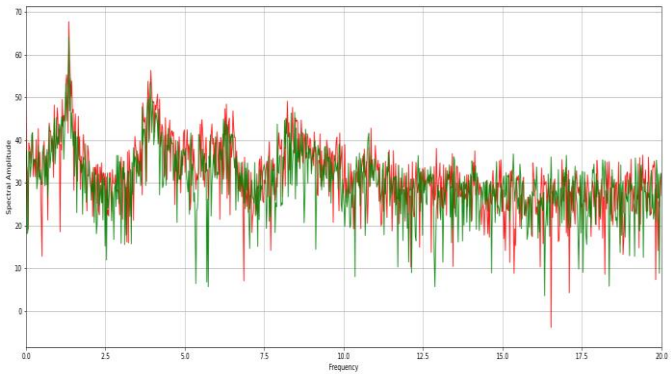


Figure 16: Frequency response

Figure 16 displays the frequency response, where the green signal corresponds to the recorded signal from Nano BLE sense and the red signal represents the predicted signal of the ANN model after training.

IV. CONCLUSION

After analyzing the SHM application, it was found that implementing ARMA models using the ANN approach is challenging due to the need for real-time sensing. Therefore, signals from numerical tool (DOF)6 data were utilized for vibration-based diagnostics [15]. The implementation of TinyML has been leveraged to execute system identification through an Autoregressive Moving Average (ARMA) model. The challenge of uniquely defining ARMA models using back propagation neural networks has been addressed. The trained ANN model's weights and biases enable the retrieval of the actual ARMA parameters, which will be presented in future developments. In the current implementation, the ANN has emulated the behavior of the ARMA model. The order of a model is determined before its parameters are estimated through coefficient of determination and the best fit information criteria and residual analysis. Insufficiently or ambiguously defined models make it impractical for the associated estimation algorithm to arrive at a convergent solution. This limits the number of possible trial models and compels the analyst to carefully select trial models. A good intuition or idea of the final model's specifications is the most reliable guarantee of finding the correct order and model specification in a reasonably short time.

REFERENCES.

[1] Stepinac, Mislav, and Mateo Gašparović. "A review of emerging technologies for an assessment of safety and seismic vulnerability and damage detection of existing masonry structures." Applied Sciences 10, no. 15 (2020): 5060.

[2] Batten, M., W. Powrie, R. Boorman, H-T. Yu, and Q. Leiper. "Use of vibrating wire strain gauges to measure loads in tubular steel props supporting deep retaining walls." Proceedings of the Institution of Civil Engineers-Geotechnical Engineering 137, no. 1 (1999): 3-13.

[3] Akpudo, Ugochukwu Ejike, and Jang-Wook Hur. "D-dCNN: A Novel Hybrid Deep Learning-Based Tool for Vibration-Based Diagnostics." Energies 14, no. 17 (2021): 5286.

[4] Fujimoto, Hiroshi. "Visual servoing of 6 dof manipulator by multirate control with depth identification." In 42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475), vol. 5, pp. 5408-5413. IEEE, 2003.

[5] Bisong, Ekaba. "Tensorflow 2.0 and keras." In Building Machine Learning and Deep Learning Models on Google Cloud Platform, pp. 347-399. Apress, Berkeley, CA, 2019.

[6] Gay, Warren. "Beginning STM32." Beginning STM32 (2018)..

[7] Neves, Ana C., Ignacio Gonzalez, John Leander, and Raid Karoumi. "Structural health monitoring of bridges: a model-free ANN-based approach to damage detection." Journal of Civil Structural Health Monitoring 7, no. 5 (2017): 689-702.

[8] Na, Dokyun, Sunjae Lee, and Doheon Lee. "Mathematical modeling of translation initiation for the estimation of its efficiency to computationally design mRNA sequences with desired expression levels in prokaryotes." BMC systems biology 4, no. 1 (2010): 1-16.

[9] Zhang, G. Peter. "Time series forecasting using a hybrid ARIMA and neural network model." Neurocomputing 50 (2003): 159-175.

[10] Alexander, David LJ, Alexander Tropsha, and David A. Winkler. "Beware of R 2: simple, unambiguous assessment of the prediction accuracy of QSAR and QSPR models." Journal of chemical information and modeling 55, no. 7 (2015): 1316-1322.

[11] Hudson, H. Malcolm, and Richard S. Larkin. "Accelerated image reconstruction using ordered subsets of projection data." IEEE transactions on medical imaging 13, no. 4 (1994): 601-609.

[12] Choi, Keunwoo, György Fazekas, Mark Sandler, and Kyunghyun Cho. "Convolutional recurrent neural networks for music classification." In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2392-2396. IEEE, 2017.

[13] Pan, Zuozhou, Zong Meng, Zijun Chen, Wenqing Gao, and Ying Shi. "A two-stage method based on extreme learning machine for predicting the remaining useful life of rolling-element bearings." Mechanical Systems and Signal Processing 144 (2020): 106899.

[14] Negash Rahmani, Amir M., Tuan Nguyen Gia, Behailu, Arman Anzanpour, Iman Azimi, Mingzhe Jiang, and Pasi Liljeberg. "Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach." Future Generation Computer Systems 78 (2018): 641-658.

[15] Matveev, V. V., and A. P. Bovsunovsky. "Vibration-based diagnostics of fatigue damage of beam-like structures." Journal of Sound and Vibration 249, no. 1 (2002): 23-40.

Van Valkenhoef, Gert, Guobing Lu, Bert de Brock, Hans Hillege, A. E. Ades, and Nicky J. Welton. "Automating network meta-analysis." Research synthesis methods 3, no. 4 (2012): 285-299.

[16] Ji, W., Zhang, X., Han, Y., & Zhou, X. " Structural Health Monitoring and Assessment of Bridges." Journal of Bridge Engineering (2018): Page numbers: 04017121.

[17] Qiang Wu, Yanmin Zhu, Houbing Song, and Qiao Ye.

" Energy-Efficient TinyML for Internet of Things: Techniques and Applications" (2022): Publisher: Springer.

[18] S. Kim, K. Lee, K. Park, and Y. Kim. " A Hybrid Machine Learning Approach for Structural Identification of Large-Scale Bridges ". Journal of Computing in Civil Engineering (2020): Page numbers: 04019088.

[19] Li, X., Li, Y., Li, L., Li, G., & Zhang, Y." A machine learning-based brain activity recognition system for wearable healthcare devices ". Journal IEEE Access (2022): Page numbers: 11711-11720.

[20] Khan, A. M., Ahmad, F., & Yousaf, M. H." A comprehensive review of internet of things architectures, technologies, and future directions " (2021). Journal IEEE Access, 9, 47361-47385.

[21] Robert H. Shumway and David S. Stoffer." Time Series Analysis and Its Applications " (2017). Publisher: Springer.