Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Especialidad en Sistemas Embebidos

Deployment of Machine Learning Algorithm to predict Battery behavior

---

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: **JESUS ALBERTO CINCO AHUMADA, JORGE ALEJANDRO FLORES TRIANA**

Director **LUIS ENRIQUE GARABITO SIORDIA**

Tlaquepaque, Jalisco. julio de 2023.

Deployment of Machine Learning Algorithm to predict Battery behavior

Jesus Alberto Cinco Ahumada
Student
ITESO
Guadalajara, Mexico
jesus.cinco@iteso.mx

Jorge Alejandro Flores Triana
Student
ITESO
Guadalajara, Mexico
jorge.flores@iteso.mx

The growth of the electric car industry has increased in recent years, along with the trend of green energy around the world. For this reason, automotive companies have invested in finding different solutions to monitor lithium batteries that power vehicles. These applications include State of Charge (SoC) and State of Health (SoH) analysis of the battery cells by monitoring key variables such as temperature, current, and voltage to predict the behavior of the system and apply preventive maintenance.

In this paper, a deep neural network using the Deep Learning MATLAB Toolbox was designed to predict the SoC from an emulated battery in Simulink. The model was then compiled and deployed in an NXP S32K344 microcontroller using the NXP Model-Based Design Toolbox. The results obtained showed a network with up to 90% accuracy and an execution time of 2.6 ms when running the core at 160 MHz.

Keywords—Battery Prediction, Embedded Systems, Machine Learning, Deployment, Embedded Coder.

I. INTRODUCTION

One of the main components that integrates an automotive vehicle is the battery, which supplies voltage and current to different systems such as the power train, telematic unit, infotainment, and body control unit, among others. In fact, the usage of this element has only increased with the implementation of electric and hybrid vehicles. However, this specific component requires a dedicated system known as Battery Management System (BMS), which monitors and controls the battery and the variables that may influence its behavior [1], such as the temperature, state of charge (SOC), and state of health (SOH). A BMS requires the implementation of several sensors to obtain the value of different variables to be monitored. Even when the implemented algorithms ensure optimal functionality of the battery and report some faults that can occur, it may be difficult to detect and report hardware faults. Therefore, some Artificial Intelligence/Machine Learning (AI/ML) algorithms have been proposed to help with the diagnosis of faults [2].

Nowadays, AI/ML algorithms are being used in different applications due to their potential, which mostly depends on how they are being implemented to ensure that the correct training has been performed over the network. An example of where these algorithms are being utilized are applications that are highly dependent on data classification. This is because AI helps to obtain the most reliable data from the set provided to the network based on a given training [3]. One of the main reasons why AI/ML algorithms are being used in BMS is because they help to process and interpret the data from different sensors. Some of these algorithms may even help to have redundancy of the measured variables when generating a model of the battery's operation [4].

The training of these AI/ML algorithms might require more processing than the typical embedded system is able to provide. For this specific problem, one solution is to deploy the algorithm with the training phase done over some simulated model using a laptop or the cloud. MATLAB and Simulink are some of the tools that are commonly employed to generate the system model, design the algorithm, and train the neural network. However, working on these platforms represents the challenge of having to migrate the entire implementation into a programming language, such as C, that can be compiled and deployed into an embedded system.

This paper describes the workflow followed to implement an AI/ML BMS algorithm in an embedded system. This process includes the design of an AI/ML algorithm that is trained with simulated data to detect anomalies on a BMS. The algorithm is then deployed in the embedded system using NXP's specific tools available on MATLAB.

The rest of the sections are organized as follows: Section II provides the design of the Machine Learning algorithm, section III shows the training with the simulated data, section IV describes in detail the method followed for the deployment, Section V explains the results obtained in the microcontroller and finally, Section VI presents the conclusions.

II. MACHINE LEARNING ALGORITHM

Artificial intelligence is the process of sharing data, information, and human intelligence with machines. The main goal is to develop self-reliant machines that can think and act like humans. Machine Learning is a discipline of computer science that uses algorithms and analytics to build predictive models capable of solving problems. Therefore, it involves the ability to adapt to new data through iterations and learn from the past to predict the future.

The workflow to solve a specific problem is contained within a defined lifecycle process. The first step is to generate a machine learning system called a Model. Then, specific data relevant to the application is used to train the Model. Finally, to

deploy the Model, the algorithm is uploaded into a device to solve the desired problem. This process is depicted in Fig. 1.

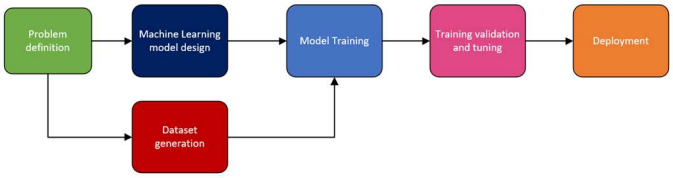

Fig. 1. Lifecycle of a Machine Learning system.

Different Machine Learning algorithms are utilized in applications such as predictive maintenance and anomaly detection depending on the analysis required over the data collected by the system. In this case, a classification algorithm is required to detect faults in the battery behavior, specifically in voltage, current, and temperature. This allows the system to easily detect when a set of data belongs to the batterie's normal operation or if it can be interpreted as a malfunction.

In Fig. 2, the overall structure for the neural network can be observed. It consists of an input layer where x represents all the input variables that are received by the network. Then, the output is represented by y_1, as it is not the output of the network but instead, it connects to another layer called hidden layer. After the hidden layer has processed all the input data from y_1, it then generates y_n, depending on the number of hidden layers in the network. This y_n is then used for the output layer. The output layer generates the expected result in the form of y with the most statistical weight used as the prediction of the Model.

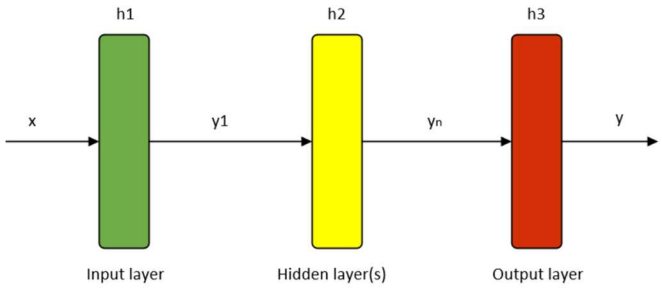

Fig. 2. Common structure of a neural network model.

The State of Charge of a battery can be defined as the ratio of charge available according to the following equation:

$$SoC(t) = \frac{Q(t)}{Q_n}, \tag{2}$$

where $Q(t)$ is the available capacity and Q_n represents the nominal capacity. However, obtaining this data tends to be difficult as the specific value for the SoC is highly dependent on the chemical properties of the battery. Therefore, it is common to study the process of SoC based on voltage, current, and temperature.

In order to define the proper model of the battery, a neural network was designed to predict the battery charge and discharge behavior. The general process is to select an activation function that makes the model follow an ideal simulation of the battery SoC. There can be more than one activation function, depending on the model, to improve its performance.

The main objective of a regression algorithm is to predict the value of a variable depending on other different inputs. There are many types of regressions depending on the system that is being analyzed. A neural network that implements this specific algorithm is usually called a convolutional neural network. A wide range of data is typically used as training material for the model and, based on the specific information, define an activation function for each layer that will integrate the model. In this case, the equations found in (1), (2), and (3), which are presented next, are the ones selected to match the data shown in section IV.

The input layer will behave as the activation function:

$$y_1 = ax^3 + bx^2 + cx + d \tag{1}$$

where y_1 represents the output from the input layer to the hidden layer, x is a three-dimensional input variable that will receive the initial voltage state, the current consumption, the time of the operation, and the temperature. Then, $a, b, c, and\ d$ are the parameters to be adjusted during the training of the neural network.

Then, the hidden layer is defined by the sigmoid function:

$$y_2 = \frac{1}{1 + e^{-y_1}}, \tag{2}$$

where y_2 represents the output from the hidden layer to the output layer, and y_1 is the input for the hidden layer coming out of the input layer.

The output layer result is obtained with the implementation of the sigmoid function as well:

$$y_3 = \frac{1}{1+e^{-y_2}}, \tag{3}$$

but in this case, it receives the output of the hidden layer with y_2 and uses it to generate y_3, which represents the output of the neural network, and it is the prediction of the system. It can also be used against the training data to verify the reliability of the result.

To evaluate the error of the output layers, the function known as the quadratic error function is utilized. It is described by:

$$\underset{a,b,c,d}{Min}\ e(y_n, y) = \frac{1}{2}(y_n - y)^2, \tag{4}$$

where y_n indicates the output of the current layer and y represents the desired value for the output.

Finally, the number of layers , the quantity of nodes, the learning parameters, and the training time need to be defined. This creates a more robust model prediction. However, the

amount of layers and neurons selected may impact the performance of the training. In this case, four neurons were selected for the input layer, as it will be receiving four different inputs in the form of time, temperature, voltage, and current. Six hidden layers are implemented with 64 neuron each. The output layer will have one neuron, as currently only the SoC is required to be obtained from the prediction of the model. The implemented neural network can be found in Fig. 3 and Fig. 4.

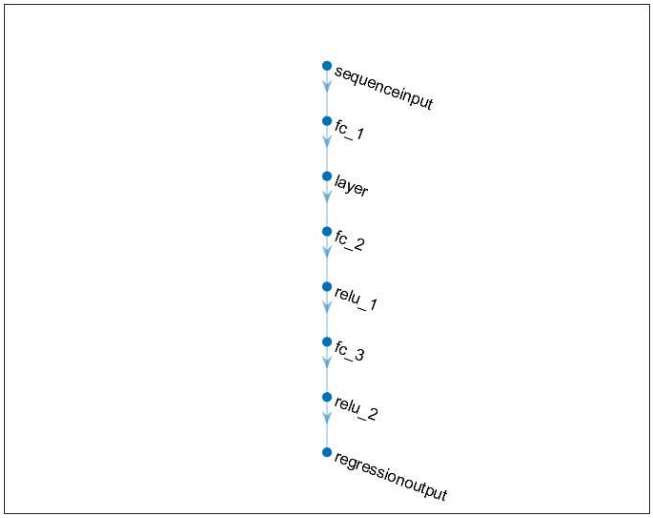

Fig. 3. Neural network implemented to describe the battery.

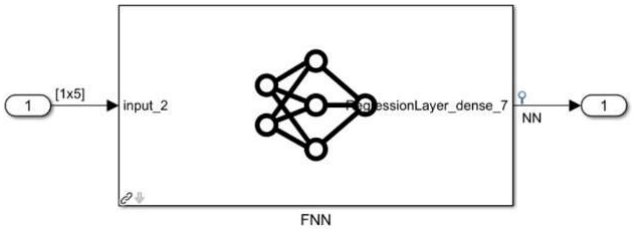

Fig. 4. Neural network block diagram in Simulink.

III. Simulated Data and Training

The BMS is responsible for performance, safe operation, and battery life of a vehicle under different environmental conditions. When designing these models, engineers develop a supervisory control algorithm that:

- Monitors cell temperature and voltage
- Estimates the SoC and SoH
- Controls the charging/discharging profile
- Limits power input supplies for thermal and overload protection
- Balances the SoC of individual cells

Simulink's simulation and modeling features enable BMS development. This includes cell equivalent circuit parameterization, control logic, automatic C-code generation, and validation. With Simulink, it is possible to emulate and design a system by

- Parameterization of equivalent circuits using tested data for accurate characterization of cell chemistry
- Modeling battery packs using electrical networks whose topology scales with the number of cells.
- Development of closed-loop control algorithms for fault detection.
- Designing state observers for parameter estimation of the SoC and SoH.
- Design of the power circuit which connects the battery pack to the control unit.

The main process for training a ML model usually starts with obtaining specific data that describes the system. Then, it is recommended to normalize the information and set the parameters in a defined scale with maximum and minimum ratings to analyze and feed into the model. Once the data is normalized, it must be separated into two groups: one that for training the network and another for validating the model. This normalization process is done using,

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}, \quad (5)$$

where x represents the current number from the set, x_{min} is the minimum value for that parameter that can be found in the dataset, and x_{max} represents the maximum value for that parameter that can be found in the dataset. The entirety of this process is graphically defined in Fig. 5.

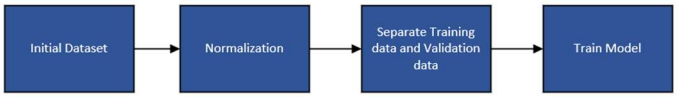

Fig. 5. Process for data conditioning and model training.

The dataset, obtained from CALCE battery team [5], to train the model defines the behavior of a battery discharging from a SOC of 100% to 0% and then being charged up again. The dataset considers a current of -0.75 Amperes for the discharge operation and a current of 0.75 Amperes for charging the battery. Both have their graphical representation in Fig. 7 and in Fig. 8 respectively.

Each stage of the battery can be analyzed independently, but there is also a set of data available describing the process from a full operation with charging and discharging stages as well. This set can be analyzed in Fig. 6.

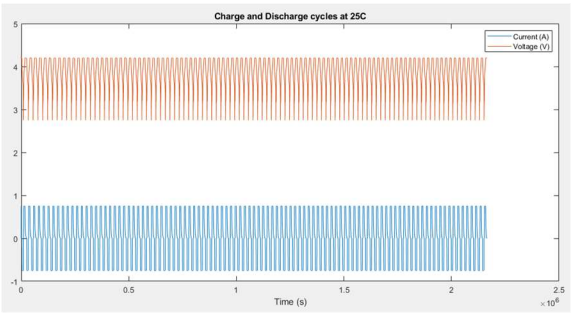

Fig. 6. Discharge and Charge cycles over time at 25C of the battery.

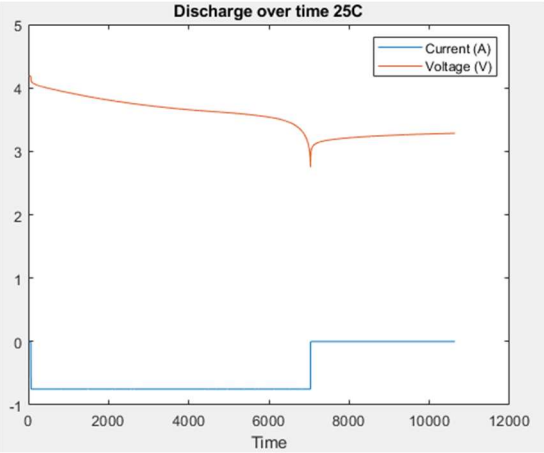

Fig. 7. Discharge data over time at 25C of the battery.

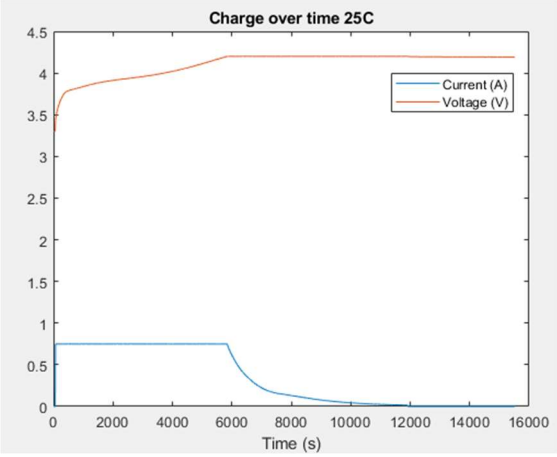

Fig. 8. Charge data over time at 25C of the battery.

With the training data obtained, it is then possible to generate the proper normalization. In this case, only the variables for the voltage and the current have been normalized, as these can be enclosed. However, this is also a time-dependent system, and it was retained to be able to predict behavior based on the amount of time and the current value. This normalization result can be observed in Fig. 9.

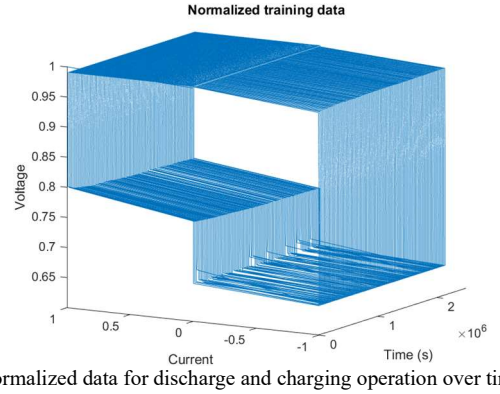

Fig. 9. Normalized data for discharge and charging operation over time.

Finally, once the data has been normalized and ordered adequately to allow the training of the model, it is required to select a specific optimization rule that will serve as the main process to follow for the neural network to train itself with the input data alongside the different cycles defined. The selected rule for this case follows the equation:

$$w_{k+1} = w_k - L_r \frac{\partial e}{\partial w}, \tag{6}$$

where w_k represents the parameters that are being optimized in the moment. In this case, it can be either a, b, c or d depending on the stage of the training. L_r is dictated by the user as it is the learning rate that is used to update the parameters, and $\frac{\partial e}{\partial w}$ is the partial derivative of the error based on the parameters that are being updated.

IV. Deployment on Microcontroller

BMS can be exercised under a variety of operating and fault conditions using Simulink before committing to hardware prototyping. It is also possible to implement control algorithms for rapid designing and deploying them in microcontrollers through the generation of C code from Simulink models. Simulink generates code from electric components, enabling real-time simulation to validate the system prior to hardware deployment.

One major task of the BMS is estimating SoC. However, traditional methods require accurate battery models that are difficult to characterize. An alternative to this is to create data-driven models of the cell using AI methods such as neural networks.

This paper shows how to use Deep Learning Toolbox, Simulink, and Embedded Coder to generate C code for AI/ML algorithms for battery SoC estimation and deploy them to an NXP S32K3 microcontroller.

Based on previous work done by McMaster University on Deep Learning workflows for battery state estimation, Embedded Coder is used to generate optimized C code from a neural network imported from TensorFlow and run it in processor-in-the-loop mode on an microcontroller [4]. The code generation workflow will feature the use of the NXP Model-Based Design Toolbox, which provides an integrated development environment and toolchain for configuring and generating all the necessary software to execute complex applications on NXP MCUs.

Embedded Coder is a tool that can be installed in MATLAB to generate code for NXP's MCUs. It is basically an extension of MATLAB Coder and Simulink Coder that allows the generation of code compliant with ANSI and ISO C standards and the direct deployment of the generated code on the specified device. However, this is not the only function that Embedded Coder provides, it also has the possibility to run in different modes:

- software in the loop (SIL), which basically consists of generating the code and running it on your computer instead of the MCU.

- process in the loop (PIL), which generates the code and uploads it to the intended MCU. Through configurable parameters, such as an UART instance, it obtains the required data to generate measurements on the generated code, such as execution time and benchmark against SIL.

Specifically, the MCU chosen was the S32K344. This is an automotive-grade general-purpose microcontroller that is currently supported within Embedded Coder. This specific device also requires the use of Real Time Drivers (RTDs), which is an Software Development Package (SDK) that allows the user to generate AUTOSAR-grade code to be implemented further in the applications. By taking advantage of this feature, it is possible to analyze the performance obtained.

Once the entire model has been simulated and verified in Simulink, the next step is to deploy it on the Microcontroller. Embedded Coder tools are used to generate a SIL simulation to validate the data from merely software running in the computer against the previously obtained information. After the results are obtained and validated, a PIL simulation is generated, and the code gets deployed directly into the microcontroller. With this process, the results are gathered via serial communication and analyzed further.

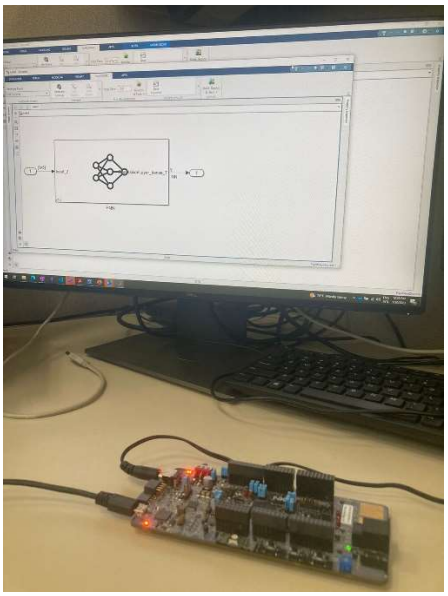

Fig. 10. S32K344 connected to the PC to deploy the generated code.

V. Results

Fig. 11 shows the results of the training step over the neural network against the real data that was used for this. Basically, this process can be defined in the SIL, thus the results yielded by the comparison of the real data against the neural network running in the computer are the training results.

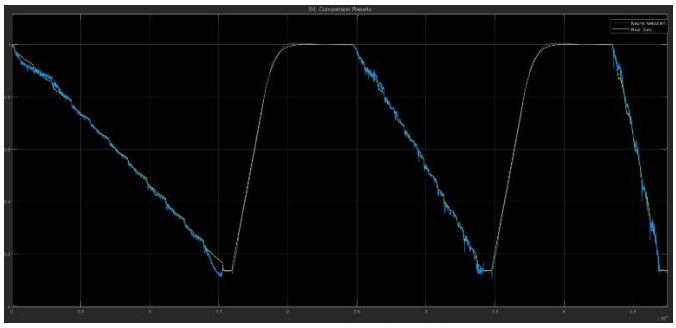

Fig.11. Neural network in SIL compared with real data.

Finally, the neural network was deployed in the microcontroller. Fig. 12, Fig. 13, Fig. 14, and Fig 15 show the proper data obtained from running the neural network on the S32K344 with Automotive-grade generated code.

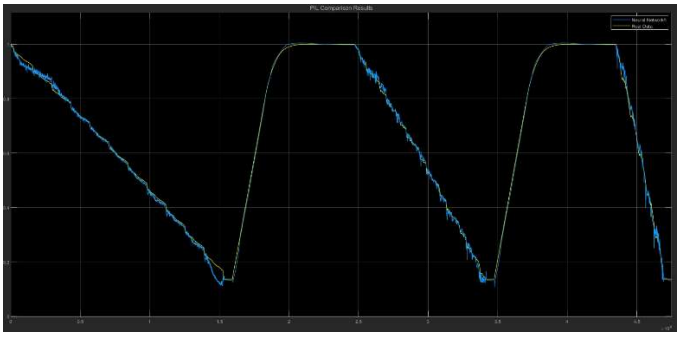

Fig. 12. Output of the neural network running in PIL.

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
nnRef_initialize	1450	1450	1450	1450	1	
nnRef_Init	300	300	300	300	1	
[-] nnRef	2381400	2360182	3150	2525	4752	
[-] nnRef_DeepLearningNetwork_predict	2378850	2357657	73300	71919	4752	
[+] nnRef_matrixMultiply1	126675	122936	10525	9455	4752	
[+] nnRef_matrixMultiply1	1096475	1082478	14150	12998	4752	
[+] nnRef_matrixMultiply1	1092050	1080325	14250	13168	4752	
nnRef_Term	550	550	550	550	1	

Fig. 13. Execution time of the functions operating in S32K344.

Section	Self Time / Caller Function	Self Time / Task	Self Time / Simulation	Function / Simulation
nnRef_initialize	100%	100%	1.2928e-05%	1.2928e-05%
nnRef_Init	100%	100%	2.6748e-06%	2.6748e-06%
[-] nnRef	0.10699%	0.10699%	0.10699%	100%
[-] nnRef_DeepLearningNetwork_predict	3.0472%	3.0472%	3.0472%	99.893%
[+] nnRef_matrixMultiply1	0.40103%	0.4006%	0.4006%	5.2083%
[+] nnRef_matrixMultiply1	0.5513%	0.55071%	0.55071%	45.864%
[+] nnRef_matrixMultiply1	0.5585%	0.5579%	0.5579%	45.773%
nnRef_Term	100%	100%	4.9039e-06%	4.9039e-06%

Fig. 14. Execution time percentages of the functions operating in S32K344.

Section	Minimum Execution Time	Average Execution Time	Maximum Execution Time	Average Self Time	Maximum Self Time	Calls	Execution Time Standard Deviation	Execution Time Variance
nnRef_initialize	1450	1450	1450	1450	1450	1	0	0
nnRef_Init	300	300	300	300	300	1	0	0
nnRef	2326150	2.3602e+06	2381400	2.5251e+03	3150	4752	5.6207e+03	3.1592e+07
nnRef_Term	550	550	550	550	550	1	0	0

Fig. 15. Tasks execution analysis in the S32K344.

VI. Conclusions

With the results obtained, the successful deployment of the neural network on the S32K344 is shown. The results showed a SoC prediction accuracy of up to 90% and a run time of 2.6 ms using the S32K344 at 160 MHz. These results also provide valuable benchmark data to consider when trying to implement AI/ML algorithms on the edge.

Typically, the industry is used to execute AI/ML models on powerful machines, such as personal laptops or the cloud. However, as technology increases, it is relevant to explore the possibility of having an embedded system with limited computing power and amount of memory, capable of running a

full AI/ML application. Thus, the main goal of building such an application on a PC, generating the C code, and implementing the model in the embedded system was also achieved.

Finally, despite the successful implementation of the model, it is also important to highlight certain concerns regarding the application because production-grade code will certainly need to be more robust, include security features, automotive-grade stacks, etc. Therefore, as next steps, it would be necessary to analyze this first integrated application to see how to integrate it to a full automotive stack.

REFERENCES

[1] G. Suciu *et al.*, «AI-based intelligent energy storage using Li-ion batteries», en *2021 12th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, Bucharest, Romania: IEEE, mar. 2021, pp. 1-5. doi: 10.1109/ATEE52255.2021.9425328.

[2] K.-T. Kim, H.-J. Lee, J.-H. Park, G. Bere, J. J. Ochoa, y T. Kim, «Artificial Intelligence-Based Hardware Fault Detection for Battery Balancing Circuits», en *2021 IEEE Energy Conversion Congress and Exposition (ECCE)*, Vancouver, BC, Canada: IEEE, oct. 2021, pp. 1387-1392. doi: 10.1109/ECCE47101.2021.9595404.

[3] K. Zidek, J. Pitel, y A. Hosovsky, «Machine learning algorithms implementation into embedded systems with web application user interface», en *2017 IEEE 21st International Conference on Intelligent Engineering Systems (INES)*, Larnaca: IEEE, oct. 2017, pp. 000077-000082. doi: 10.1109/INES.2017.8118532.

[4] M. Naguib, P. Kollmeyer, y A. Emadi, «Application of Deep Neural Networks for Lithium-Ion Battery Surface Temperature Estimation Under Driving and Fast Charge Conditions», *IEEE Trans. Transp. Electrific.*, vol. 9, n.º 1, pp. 1153-1165, mar. 2023, doi: 10.1109/TTE.2022.3200225.

[5] Saurabh Saxena, Christopher Hendricks and Michael Pecht, «Cycle Life Testing and Modeling of Graphite/ LiCoO2 cells under different state of charge ranges», *Journal of Power Sources*, 327 (2016), pp.394-400, 2016.